CAP 6619 - Deep Learning

Project 4

Image classification using the CIFAR-10 dataset

Richard Acs (Z23536011) and Matthew Acs (Z23536012)

Inspired by:

https://www.kaggle.com/c/cifar-10

https://keras.io/examples/vision/metric_learning/

https://www.kaggle.com/roblexnana/cifar10-with-cnn-for-beginer

Setup

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
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 os

import numpy as np

import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import itertools

%matplotlib inline

import random

import tensorflow as tf
from collections import defaultdict
from PIL import Image
```

```
from sklearn.metrics import ConfusionMatrixDisplay
from tensorflow import keras
from tensorflow.keras import layers
```

Part 1: Load dataset

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the image data
x_train = x_train.astype("float32") / 255.0
y_train = np.squeeze(y_train)
x_test = x_test.astype("float32") / 255.0
y_test = np.squeeze(y_test)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
170508288/170498071 [==============================] - 3s 0us/step
```

Part 2: Explore dataset

```
# Number of samples
print("Number of training sample: ",y_train.shape[0])
print("Number of test samples: ", y_test.shape[0])
```

```
Number of training sample:  50000
Number of test samples:  10000
```

```
# Number of classes
num_classes = max(y_test)+1
print(num_classes)
```

```
10
```

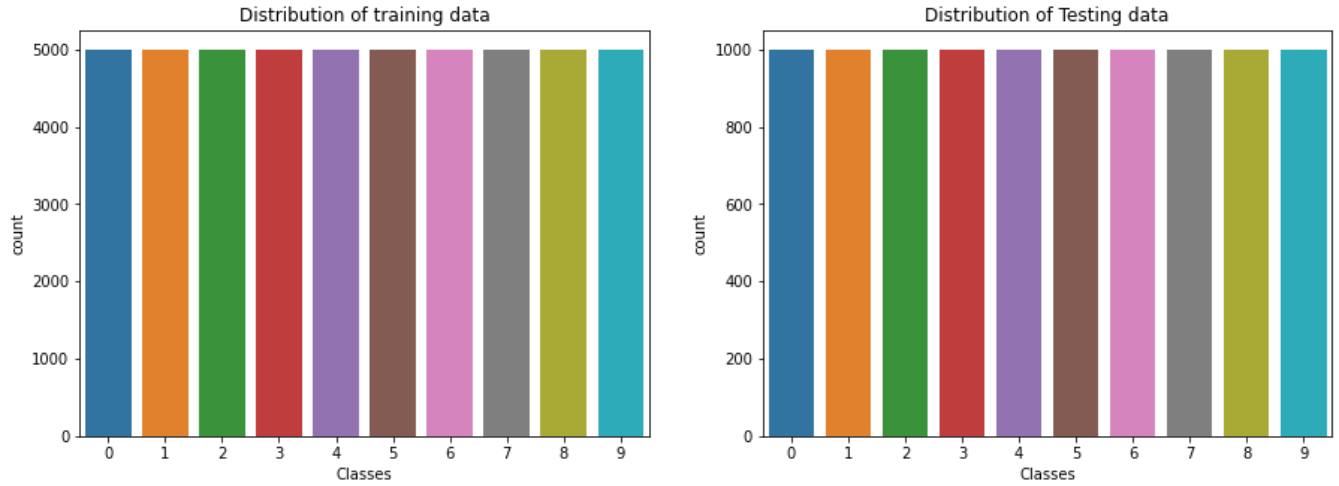
```
# Shape of image data
print(x_train.shape)
print(x_test.shape)
```

```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

```
# Distribution of classes in training samples

fig, axs = plt.subplots(1,2,figsize=(15,5))
# Count plot for training set
sns.countplot(y_train.ravel(), ax=axs[0])
```

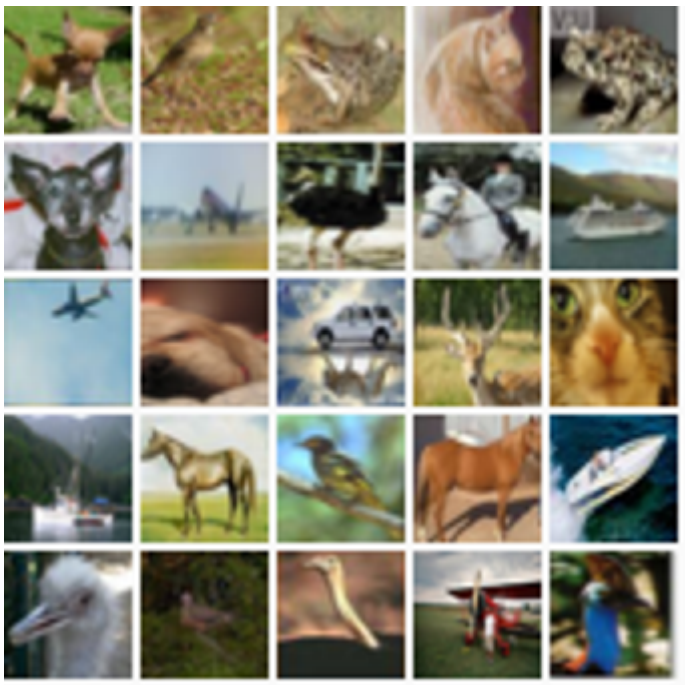
```
axs[0].set_title('Distribution of training data')
axs[0].set_xlabel('Classes')
# Count plot for testing set
sns.countplot(y_test.ravel(), ax=axs[1])
axs[1].set_title('Distribution of Testing data')
axs[1].set_xlabel('Classes')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
  FutureWarning
```




```
# Show collage of 25 (randomly selected) images

height_width = 32


def show_collage(examples):
    box_size = height_width + 2
    num_rows, num_cols = examples.shape[:2]

    collage = Image.new(
        mode="RGB",
        size=(num_cols * box_size, num_rows * box_size),
        color=(250, 250, 250),
    )
    for row_idx in range(num_rows):
        for col_idx in range(num_cols):
            array = (np.array(examples[row_idx, col_idx]) * 255).astype(np.uint8)
            collage.paste(
                Image.fromarray(array), (col_idx * box_size, row_idx * box_size)
            )
```

```
    # Double size for visualisation.
    collage = collage.resize((2 * num_cols * box_size, 2 * num_rows * box_size))
    return collage


# Show a collage of 5x5 random images.
sample_idxs = np.random.randint(0, 50000, size=(5, 5))
examples = x_train[sample_idxs]
show_collage(examples)
```



```
idx = y_train[0]
print(idx)
```

```
6
```

```
labels = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'S
```

```
labels[idx]
```

```
'Frog'
```

```
plt.imshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7fafed3783d0>
```


```
# Convert class vectors to binary class matrices.
# This is called one-hot encoding.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
y_train[0]
```

```
array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

Part 3: Build your 1st model

In this case we will start by using a convolutional neural network (CNN) built from scratch.

```
batch_size = 128
epochs = 40
data_augmentation = False
```

```
#define the convnet
model = Sequential()
# CONV => RELU => CONV => RELU => POOL => DROPOUT
model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# CONV => RELU => CONV => RELU => POOL => DROPOUT
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# FLATTEN => DENSE => RELU => DROPOUT
model.add(Flatten())
model.add(Dense(512))
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))
# a softmax classifier
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 activation (Activation)     (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 30, 30, 32)        9248

 activation_1 (Activation)   (None, 30, 30, 32)        0

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 dropout (Dropout)           (None, 15, 15, 32)        0

 conv2d_2 (Conv2D)           (None, 15, 15, 64)        18496

 activation_2 (Activation)   (None, 15, 15, 64)        0

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        36928

 activation_3 (Activation)   (None, 13, 13, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 6, 6, 64)          0

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

 activation_4 (Activation)   (None, 512)               0

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

 activation_5 (Activation)   (None, 10)                0

=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

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

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

Part 4: Train your first model

```
history = model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
```

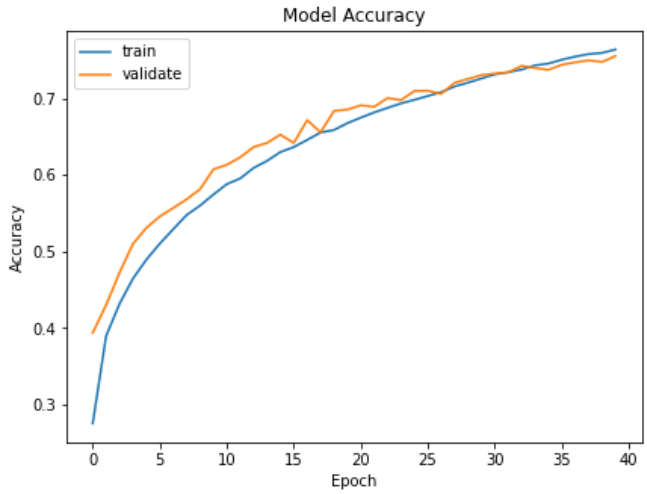
```
Epoch 12/40
391/391 [==============================] - 5s 13ms/step - loss: 1.1362 - accurac
Epoch 13/40
391/391 [==============================] - 5s 12ms/step - loss: 1.1058 - accurac
Epoch 14/40
391/391 [==============================] - 5s 13ms/step - loss: 1.0799 - accurac
Epoch 15/40
391/391 [==============================] - 5s 13ms/step - loss: 1.0517 - accurac
Epoch 16/40
391/391 [==============================] - 5s 12ms/step - loss: 1.0307 - accurac
Epoch 17/40
391/391 [==============================] - 5s 12ms/step - loss: 1.0093 - accurac
Epoch 18/40
391/391 [==============================] - 5s 12ms/step - loss: 0.9851 - accurac
Epoch 19/40
391/391 [==============================] - 5s 13ms/step - loss: 0.9701 - accurac
Epoch 20/40
391/391 [==============================] - 5s 12ms/step - loss: 0.9462 - accurac
Epoch 21/40
391/391 [==============================] - 5s 13ms/step - loss: 0.9287 - accurac
Epoch 22/40
391/391 [==============================] - 5s 13ms/step - loss: 0.9088 - accurac
Epoch 23/40
391/391 [==============================] - 5s 13ms/step - loss: 0.8944 - accurac
Epoch 24/40
391/391 [==============================] - 5s 13ms/step - loss: 0.8804 - accurac
Epoch 25/40
391/391 [==============================] - 5s 12ms/step - loss: 0.8640 - accurac
Epoch 26/40
391/391 [==============================] - 5s 13ms/step - loss: 0.8485 - accurac
Epoch 27/40
391/391 [==============================] - 5s 13ms/step - loss: 0.8341 - accurac
Epoch 28/40
391/391 [==============================] - 5s 13ms/step - loss: 0.8161 - accurac
Epoch 29/40
391/391 [==============================] - 5s 13ms/step - loss: 0.8029 - accurac
Epoch 30/40
```

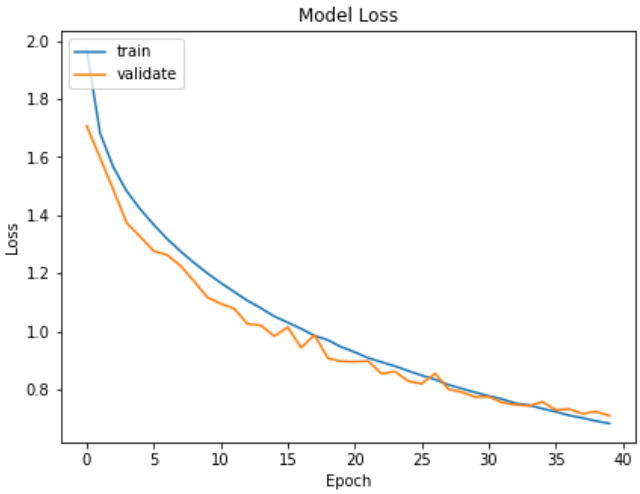
```
Epoch 30/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7898 - accurac
Epoch 31/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7777 - accurac
Epoch 32/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7673 - accurac
Epoch 33/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7523 - accurac
Epoch 34/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7463 - accurac
Epoch 35/40
391/391 [==============================] - 5s 14ms/step - loss: 0.7341 - accurac
Epoch 36/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7235 - accurac
Epoch 37/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7108 - accurac
Epoch 38/40
391/391 [==============================] - 5s 13ms/step - loss: 0.7021 - accurac
Epoch 39/40
391/391 [==============================] - 5s 13ms/step - loss: 0.6916 - accurac
Epoch 40/40
391/391 [==============================] - 5s 13ms/step - loss: 0.6831 - accurac
```

```
def plotmodelhistory(history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(history.history['accuracy'])
    axs[0].plot(history.history['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].legend(['train', 'validate'], loc='upper left')
    # summarize history for loss
    axs[1].plot(history.history['loss'])
    axs[1].plot(history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].legend(['train', 'validate'], loc='upper left')
    plt.show()

# list all data in history
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plotmodelhistory(history)
```





Part 5: Evaluate your 1st model

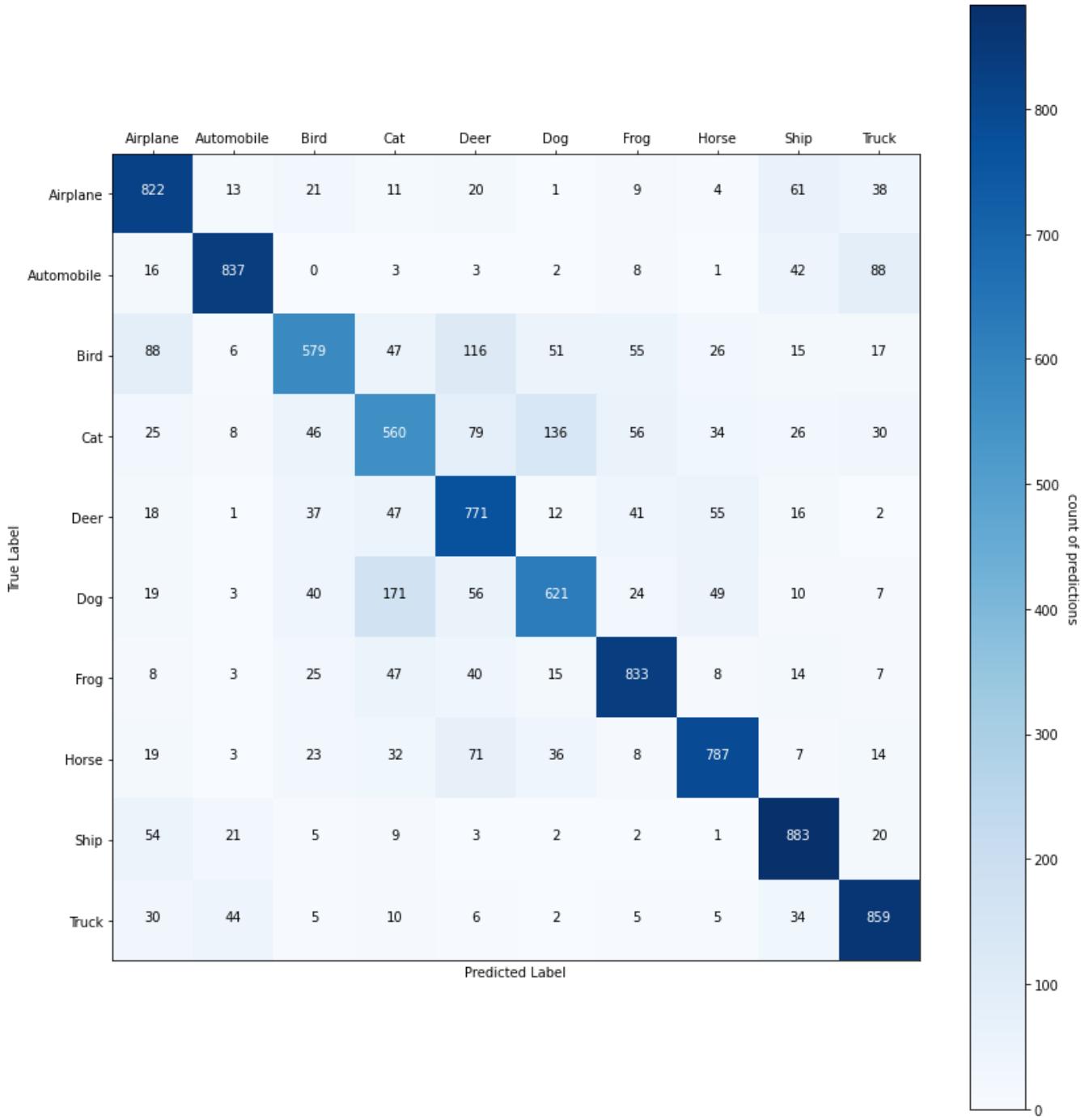
```
# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# make prediction.
pred = model.predict(x_test)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.7098 - accuracy
Test loss: 0.7097917795181274
Test accuracy: 0.7552000284194946
```

```
def heatmap(data, row_labels, col_labels, ax=None, cbar_kw={}, cbarlabel="", **kwargs
    """
    Create a heatmap from a numpy array and two lists of labels.
    """
    if not ax:
        ax = plt.gca()

    # Plot the heatmap
    im = ax.imshow(data, **kwargs)

    # Create colorbar
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

    # Let the horizontal axes labeling appear on top.
    ax.tick_params(top=True, bottom=False,
                   labeltop=True, labelbottom=False)
    # We want to show all ticks...
    ax.set_xticks(np.arange(data.shape[1]))
    ax.set_yticks(np.arange(data.shape[0]))
    # ... and label them with the respective list entries.
```

```
    ax.set_xticklabels(col_labels)
    ax.set_yticklabels(row_labels)

    ax.set_xlabel('Predicted Label')
    ax.set_ylabel('True Label')

    return im, cbar

def annotate_heatmap(im, data=None, fmt="d", threshold=None):
    """
    A function to annotate a heatmap.
    """
    # Change the text's color depending on the data.
    texts = []
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            text = im.axes.text(j, i, format(data[i, j], fmt), horizontalalignment="c
                                color="white" if data[i, j] > thresh else "black")
            texts.append(text)

    return texts
```

```
# Plot confusion matrix

# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(pred, axis=1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis=1)
# Errors are difference between predicted labels and true labels
errors = (Y_pred_classes - Y_true != 0)

Y_pred_classes_errors = Y_pred_classes[errors]
Y_pred_errors = pred[errors]
Y_true_errors = Y_true[errors]
X_test_errors = x_test[errors]

cm = confusion_matrix(Y_true, Y_pred_classes)
thresh = cm.max() / 2.

fig, ax = plt.subplots(figsize=(12,12))
im, cbar = heatmap(cm, labels, labels, ax=ax,
                   cmap=plt.cm.Blues, cbarlabel="count of predictions")
texts = annotate_heatmap(im, data=cm, threshold=thresh)

fig.tight_layout()
plt.show()
```

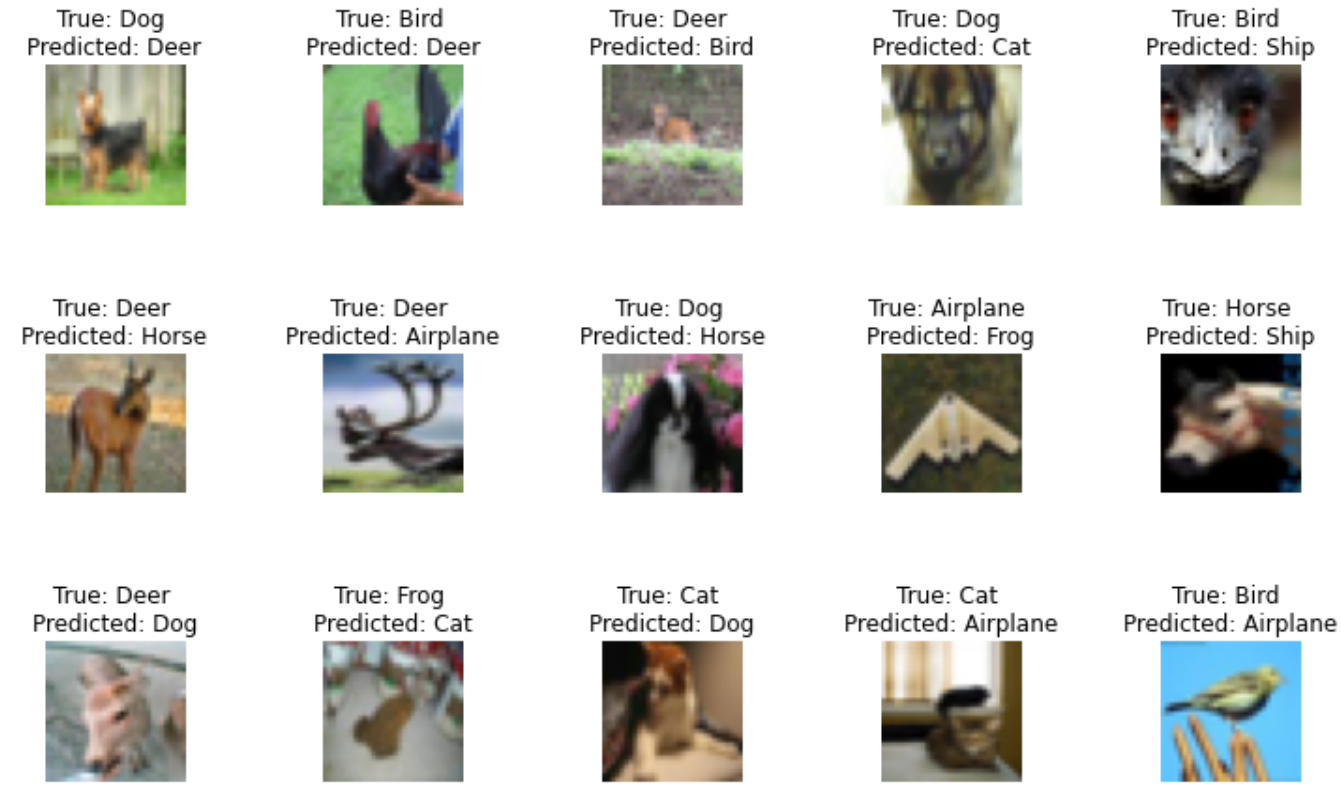



```
print(classification_report(Y_true, Y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.75      0.82      0.78      1000
           1       0.89      0.84      0.86      1000
           2       0.74      0.58      0.65      1000
           3       0.60      0.56      0.58      1000
           4       0.66      0.77      0.71      1000
```

```
              5       0.71      0.62      0.66      1000
              6       0.80      0.83      0.82      1000
              7       0.81      0.79      0.80      1000
              8       0.80      0.88      0.84      1000
              9       0.79      0.86      0.83      1000

       accuracy                           0.76     10000
      macro avg       0.75      0.76      0.75     10000
   weighted avg       0.75      0.76      0.75     10000
```

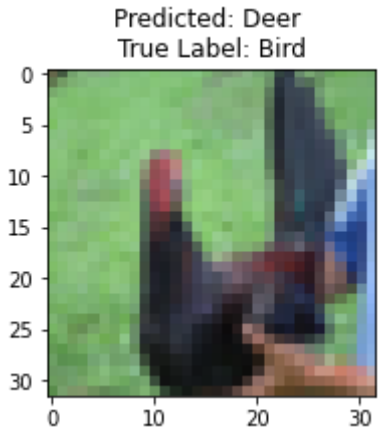
```
# Inspect errors
R = 3
C = 5
fig, axes = plt.subplots(R, C, figsize=(12,8))
axes = axes.ravel()

misclassified_idx = np.where(Y_pred_classes != Y_true)[0]
for i in np.arange(0, R*C):
    axes[i].imshow(x_test[misclassified_idx[i]])
    axes[i].set_title("True: %s \nPredicted: %s" % (labels[Y_true[misclassified_idx[i
                                                   labels[Y_pred_classes[misclassified
    axes[i].axis('off')
    plt.subplots_adjust(wspace=1)
```




```
def show_test(number):
    fig = plt.figure(figsize = (3,3))
```

```
    test_image = np.expand_dims(x_test[number], axis=0)
    predict_x=model.predict(test_image)
    test_result=np.argmax(predict_x,axis=1)
    plt.imshow(x_test[number])
    dict_key = test_result[0]
    plt.title("Predicted: {} \nTrue Label: {}".format(labels[dict_key],
                                                      labels[Y_true[number]]))
```

```
show_test(25)
```



Part 6: Transfer Learning

Transfer Learning: setup

```
from keras import Sequential
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import VGG19
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau
from keras.layers import Flatten,Dense,BatchNormalization,Activation,Dropout
from tensorflow.keras.utils import to_categorical
```

```
#Import dataset (again)
(x_train,y_train),(x_test,y_test)=cifar10.load_data()
```

```
x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=.3)
```

```
#Print the dimensions of the datasets to make sure everything's kosher

print((x_train.shape,y_train.shape))
```

```
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

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

```
#One hot encode the labels.Since we have 10 classes we should expect the shape[1] of

y_train=to_categorical(y_train)
y_val=to_categorical(y_val)
y_test=to_categorical(y_test)
```

```
# Lets print the dimensions one more time to see if things changed the way we expecte

print((x_train.shape,y_train.shape))
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

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

Attempt #1: using ResNet50 as a base model

Learn more about ResNet50 at: https://www.kaggle.com/keras/resnet50

```
base_model_1 = ResNet50(include_top=False,
                        weights='imagenet',
                        input_shape=(32,32,3),
                        classes=y_train.shape[1])

model_1=Sequential()
#Add the Dense layers along with activation and batch normalization
model_1.add(base_model_1)
model_1.add(Flatten())

#Add the Dense layers along with activation and batch normalization
model_1.add(Dense(4000,activation=('relu'),input_dim=512))
model_1.add(Dense(2000,activation=('relu')))
model_1.add(Dense(1000,activation=('relu')))
model_1.add(Dense(500,activation=('relu')))
model_1.add(Dense(10,activation=('softmax'))) #This is the classification layer
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

```
batch_size= 128
epochs=10
```

```
learn_rate=.003

sgd=SGD(learning_rate=learn_rate,momentum=.9,nesterov=False)
adam=Adam(learning_rate=learn_rate)
```

```
# Compile the model
model_1.compile(optimizer=sgd,
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
# Train the model
history_1 = model_1.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose = 1,
              validation_data=(x_val, y_val),
              shuffle=True)
```

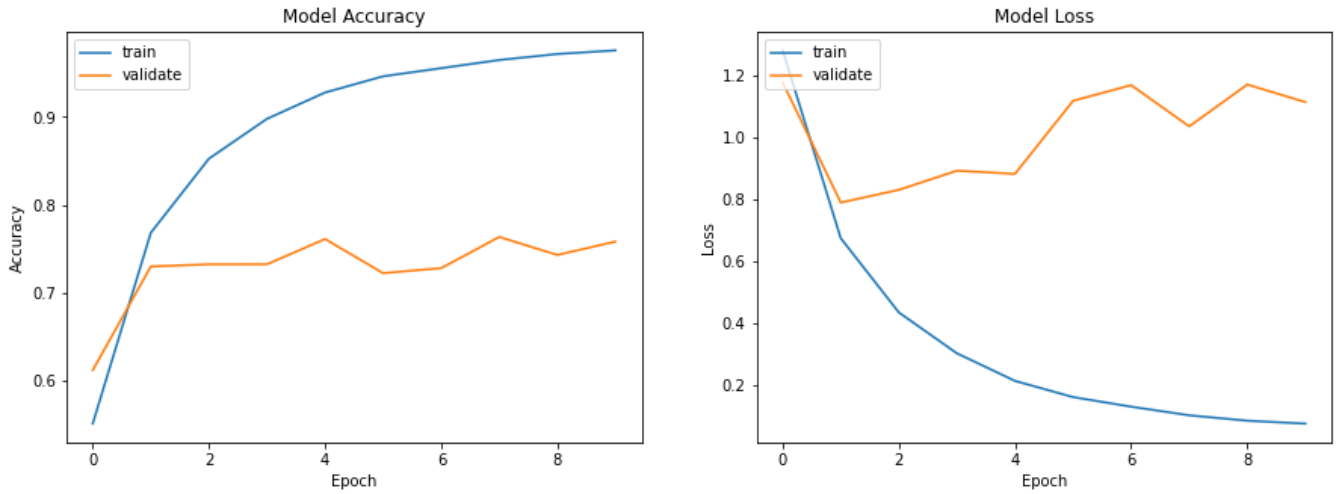
```
Epoch 1/10
274/274 [==============================] - 29s 82ms/step - loss: 1.2794 - accura
Epoch 2/10
274/274 [==============================] - 20s 73ms/step - loss: 0.6728 - accura
Epoch 3/10
274/274 [==============================] - 21s 75ms/step - loss: 0.4331 - accura
Epoch 4/10
274/274 [==============================] - 20s 75ms/step - loss: 0.3011 - accura
Epoch 5/10
274/274 [==============================] - 21s 75ms/step - loss: 0.2121 - accura
Epoch 6/10
274/274 [==============================] - 21s 75ms/step - loss: 0.1597 - accura
Epoch 7/10
274/274 [==============================] - 21s 75ms/step - loss: 0.1288 - accura
Epoch 8/10
274/274 [==============================] - 21s 75ms/step - loss: 0.1009 - accura
Epoch 9/10
274/274 [==============================] - 21s 75ms/step - loss: 0.0835 - accura
Epoch 10/10
274/274 [==============================] - 21s 75ms/step - loss: 0.0741 - accura
```

```
def plotmodelhistory(history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(history.history['accuracy'])
    axs[0].plot(history.history['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].legend(['train', 'validate'], loc='upper left')
    # summarize history for loss
    axs[1].plot(history.history['loss'])
```

```
    axs[1].plot(history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].legend(['train', 'validate'], loc='upper left')
    plt.show()

# list all data in history
print(history_1.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plotmodelhistory(history_1)
```




```
# Score trained model.
scores = model_1.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# make prediction.
pred = model_1.predict(x_test)
```

```
313/313 [==============================] - 5s 16ms/step - loss: 1.1835 - accurac
Test loss: 1.183546781539917
Test accuracy: 0.748199999332428
```

```
  def plot_confusion_matrix(y_true, y_pred, classes,
                            normalize=False,
                            title=None,
                            cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
```

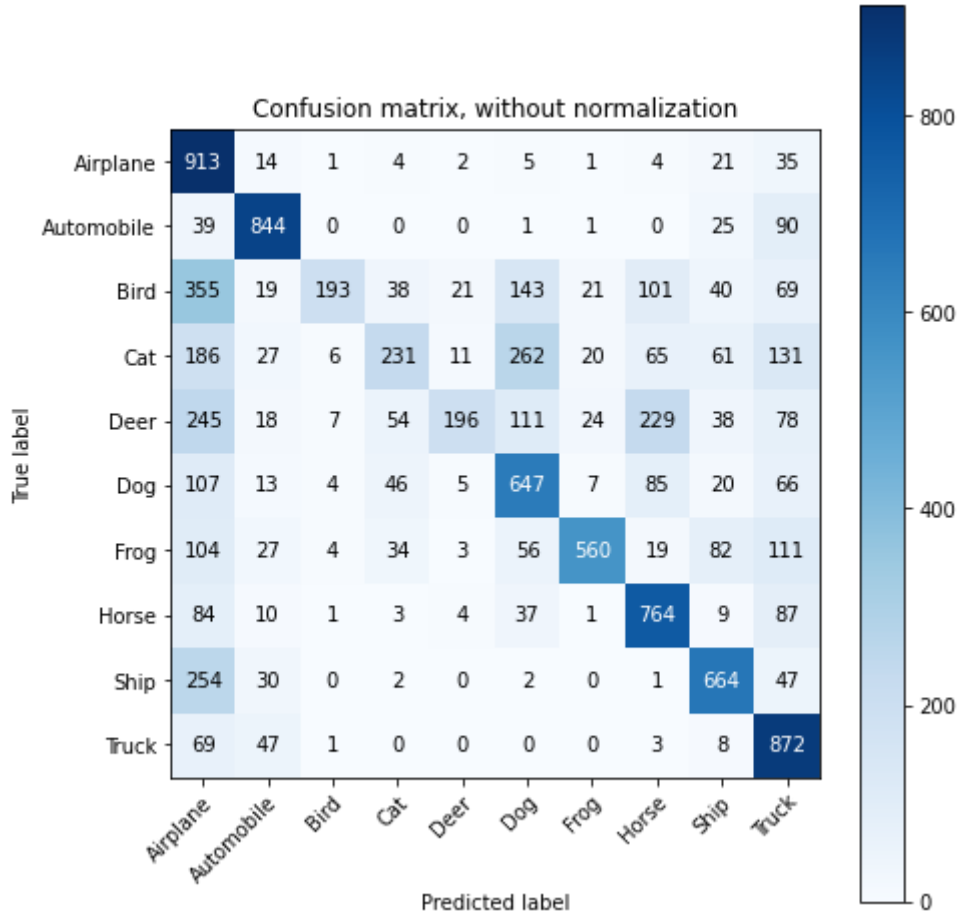
```
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

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

#     print(cm)

    fig, ax = plt.subplots(figsize=(7,7))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")
    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)
```

```
predict_x=model.predict(x_test)
y_pred=np.argmax(predict_x,axis=1)
y_true=np.argmax(y_test,axis=1)
```

```
#Compute the confusion matrix
confusion_mtx=confusion_matrix(y_true,y_pred)
```

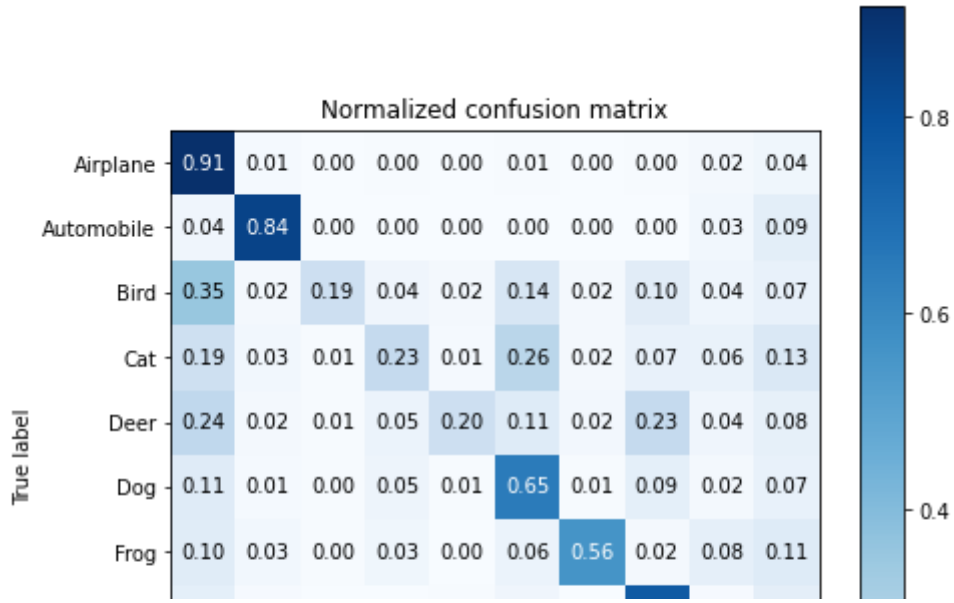
```
# Plot non-normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes=labels,
                      title='Confusion matrix, without normalization')
```

```
Confusion matrix, without normalization
<matplotlib.axes._subplots.AxesSubplot at 0x7fae9d72af50>
```




```
# Plot normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes=labels, normalize=True,
                      title='Normalized confusion matrix')
# plt.show()
```

```
Normalized confusion matrix
<matplotlib.axes._subplots.AxesSubplot at 0x7fae9d56ff90>
```



Attempt #2: Your turn!

Write code to build, train, and evaluate *another* transfer learning model.

Here are a few things you could do:

- Use a different base model, e.g., VGG19 (see https://www.kaggle.com/keras/vgg19)
- Add Dropout layers
- Use data augmentation
- Change optimizer
- Change other hyperparameters (learning rate, batch size, etc.)

SAMPLE SOLUTION

The code below uses the same base model and simply changes optimizer and other hyperparameters.

```
base_model_4 = ResNet50(include_top=False,
                        weights='imagenet',
                        input_shape=(32,32,3),
                        classes=y_train.shape[1])

model_4=Sequential()
#Add the Dense layers along with activation and batch normalization
model_4.add(base_model_4)
model_4.add(Flatten())

#Add the Dense layers along with activation and batch normalization
model_4.add(Dense(4000,activation=('relu'),input_dim=512))
model_4.add(Dense(2000,activation=('relu')))
```

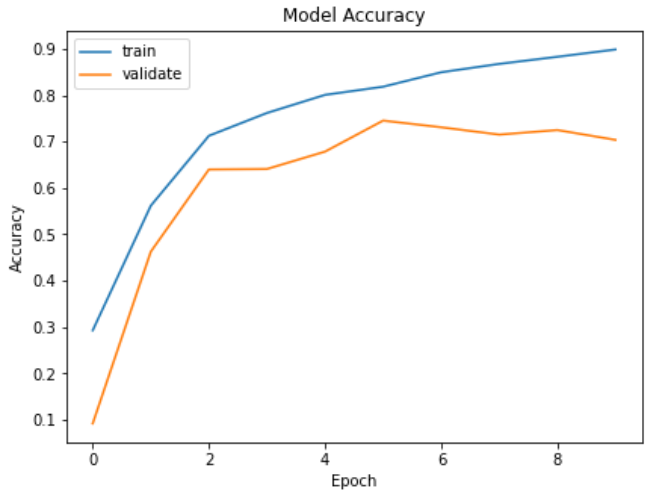
```
model_4.add(Dense(1000,activation=('relu')))
model_4.add(Dense(500,activation=('relu')))
model_4.add(Dense(10,activation=('softmax'))) #This is the classification layer
```

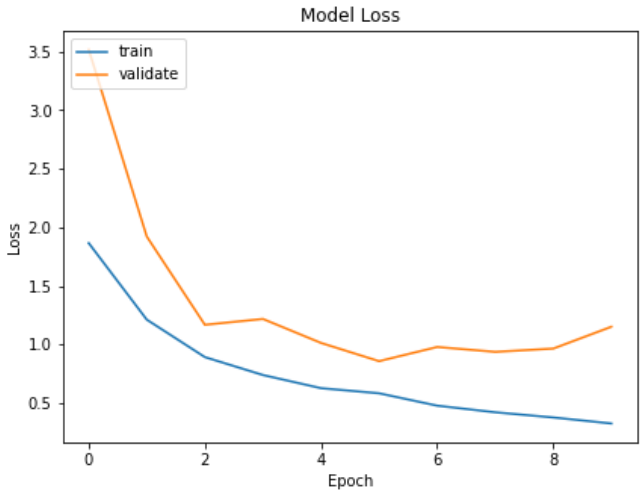
```
# Compile the model
batch_size= 128
epochs=10
learn_rate=.001

sgd=SGD(learning_rate=learn_rate,momentum=.9,nesterov=False)
adam=Adam(learning_rate=learn_rate)

model_4.compile(optimizer=adam,
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
# Train the model
history_4 = model_4.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_val, y_val),
              shuffle=True)
```

```
Epoch 1/10
274/274 [==============================] - 27s 79ms/step - loss: 1.8658 - accura
Epoch 2/10
274/274 [==============================] - 21s 77ms/step - loss: 1.2126 - accura
Epoch 3/10
274/274 [==============================] - 21s 77ms/step - loss: 0.8931 - accura
Epoch 4/10
274/274 [==============================] - 21s 76ms/step - loss: 0.7401 - accura
Epoch 5/10
274/274 [==============================] - 21s 77ms/step - loss: 0.6279 - accura
Epoch 6/10
274/274 [==============================] - 21s 76ms/step - loss: 0.5843 - accura
Epoch 7/10
274/274 [==============================] - 21s 77ms/step - loss: 0.4787 - accura
Epoch 8/10
274/274 [==============================] - 21s 76ms/step - loss: 0.4222 - accura
Epoch 9/10
274/274 [==============================] - 21s 76ms/step - loss: 0.3780 - accura
Epoch 10/10
274/274 [==============================] - 21s 77ms/step - loss: 0.3267 - accura
```

```
plotmodelhistory(history_4)
```



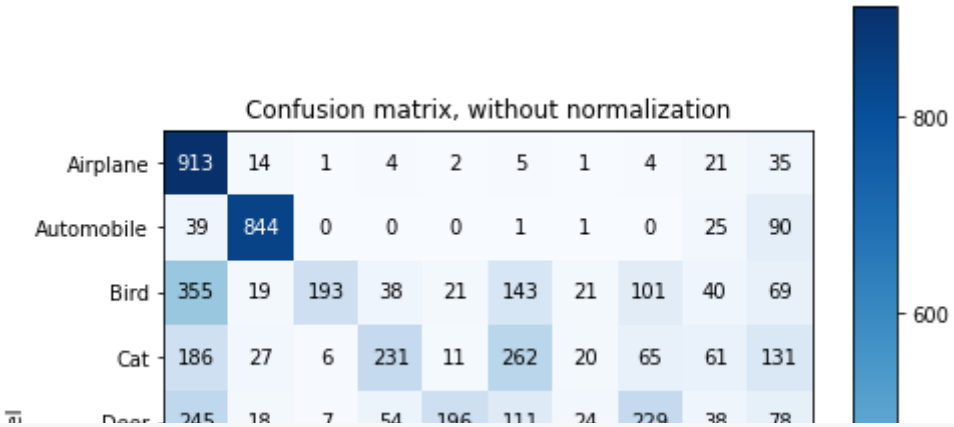



```
# Score trained model.
scores = model_4.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# make prediction.
pred = model_4.predict(x_test)
```

```
313/313 [==============================] - 5s 15ms/step - loss: 1.1542 - accurac
Test loss: 1.1542142629623413
Test accuracy: 0.7020000219345093
```

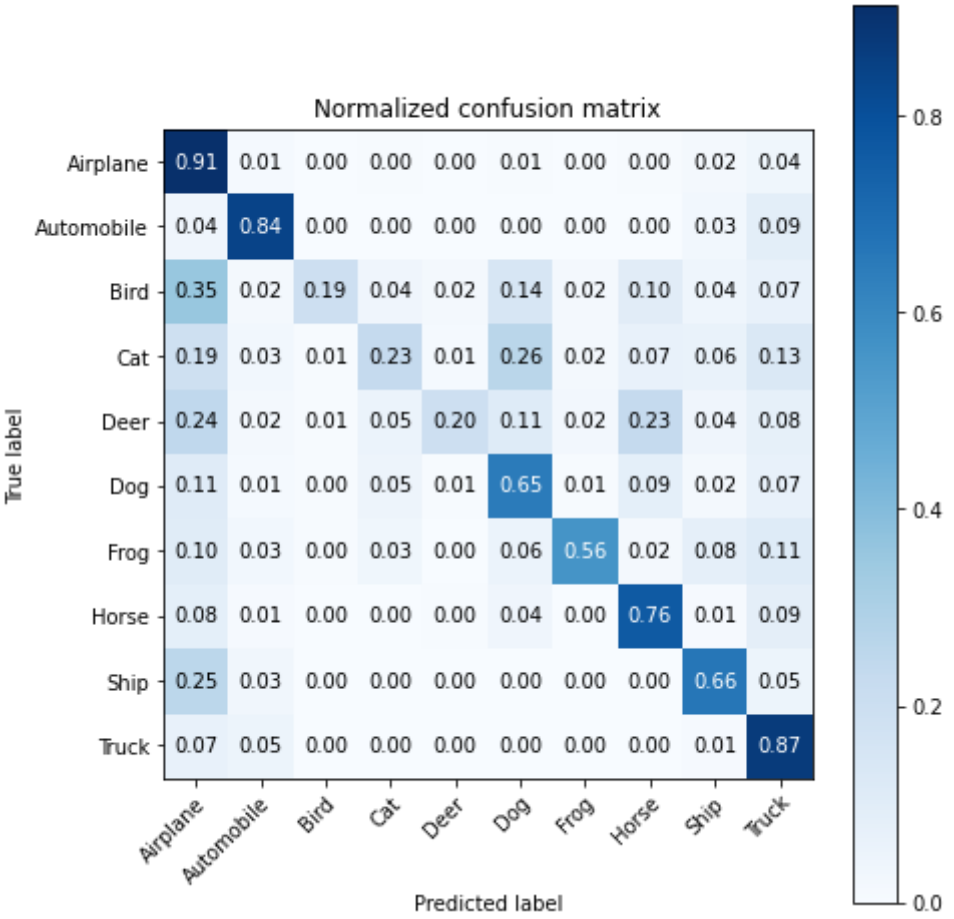
```
predict_x=model.predict(x_test)
y_pred=np.argmax(predict_x,axis=1)
y_true=np.argmax(y_test,axis=1)

#Compute the confusion matrix
confusion_mtx=confusion_matrix(y_true,y_pred)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes=labels,
                      title='Confusion matrix, without normalization')
```

```
Confusion matrix, without normalization
<matplotlib.axes._subplots.AxesSubplot at 0x7fae7b098b90>
```




```
# Plot normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes=labels, normalize=True,
                      title='Normalized confusion matrix')
# plt.show()
```

```
Normalized confusion matrix
<matplotlib.axes._subplots.AxesSubplot at 0x7fae7a771b10>
```



Attempt #3: Adding data augmentation and dropout

```
base_model_5 = ResNet50(include_top=False,
                        weights='imagenet',
                        input_shape=(32,32,3),
                        classes=y_train.shape[1])

data_augmentation = keras.Sequential(
    [
     layers.RandomFlip("horizontal"),
     layers.RandomRotation(0.1),
     layers.RandomZoom(0.2),
    ]
)

model_5=Sequential()
#Add the Dense layers along with activation and batch normalization
model_5.add(data_augmentation)
model_5.add(base_model_5)
model_5.add(Flatten())

#Add the Dense layers along with activation and batch normalization
model_5.add(Dense(4000,activation=('relu'),input_dim=512))
model_5.add(Dense(2000,activation=('relu')))
model_5.add(Dense(1000,activation=('relu')))
model_5.add(Dense(500,activation=('relu')))
model_5.add(Dropout(0.5))
model_5.add(Dense(10,activation=('softmax'))) #This is the classification layer
```

```
batch_size= 128
epochs=50
learn_rate=.003

sgd=SGD(learning_rate=learn_rate,momentum=.9,nesterov=False)
adam=Adam(learning_rate=learn_rate)
```

```
# Compile the model
model_5.compile(optimizer=sgd,
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
callbacks = [
             keras.callbacks.ModelCheckpoint(
                 filepath="best.keras",
                 save_best_only = True,
                 monitor = "val_loss")
]

# Train the model
history_5 = model_5.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
```

```
            verbose = 1,
            validation_data=(x_val, y_val),
            shuffle=True,
            callbacks=callbacks)
```

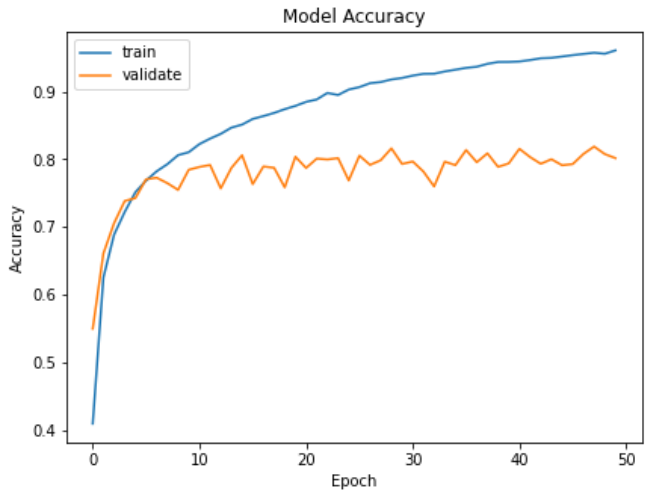
```
====================] - 30s 86ms/step - loss: 1.6485 - accuracy: 0.4093 - val_los

====================] - 22s 81ms/step - loss: 1.0926 - accuracy: 0.6252 - val_los

====================] - 22s 82ms/step - loss: 0.9178 - accuracy: 0.6885 - val_los

====================] - 23s 83ms/step - loss: 0.8175 - accuracy: 0.7222 - val_los

====================] - 23s 83ms/step - loss: 0.7381 - accuracy: 0.7513 - val_los

====================] - 22s 82ms/step - loss: 0.6807 - accuracy: 0.7685 - val_los

====================] - 23s 82ms/step - loss: 0.6424 - accuracy: 0.7823 - val_los

====================] - 21s 78ms/step - loss: 0.6067 - accuracy: 0.7928 - val_los

====================] - 21s 77ms/step - loss: 0.5741 - accuracy: 0.8061 - val_los

====================] - 22s 82ms/step - loss: 0.5565 - accuracy: 0.8104 - val_los

====================] - 21s 78ms/step - loss: 0.5167 - accuracy: 0.8224 - val_los

====================] - 23s 84ms/step - loss: 0.4978 - accuracy: 0.8305 - val_los

====================] - 21s 78ms/step - loss: 0.4723 - accuracy: 0.8375 - val_los

====================] - 21s 78ms/step - loss: 0.4468 - accuracy: 0.8467 - val_los

====================] - 22s 82ms/step - loss: 0.4358 - accuracy: 0.8512 - val_los

====================] - 21s 78ms/step - loss: 0.4119 - accuracy: 0.8595 - val_los

====================] - 21s 78ms/step - loss: 0.3999 - accuracy: 0.8638 - val_los

====================] - 21s 78ms/step - loss: 0.3820 - accuracy: 0.8684 - val_los

====================] - 21s 78ms/step - loss: 0.3683 - accuracy: 0.8740 - val_los

====================] - 21s 77ms/step - loss: 0.3485 - accuracy: 0.8788 - val_los

====================] - 21s 77ms/step - loss: 0.3382 - accuracy: 0.8848 - val_los

====================] - 21s 78ms/step - loss: 0.3251 - accuracy: 0.8883 - val_los

====================] - 21s 78ms/step - loss: 0.3021 - accuracy: 0.8980 - val_los

====================] - 21s 77ms/step - loss: 0.3050 - accuracy: 0.8949 - val_los

====================] - 22s 79ms/step - loss: 0.2847 - accuracy: 0.9030 - val_los
```

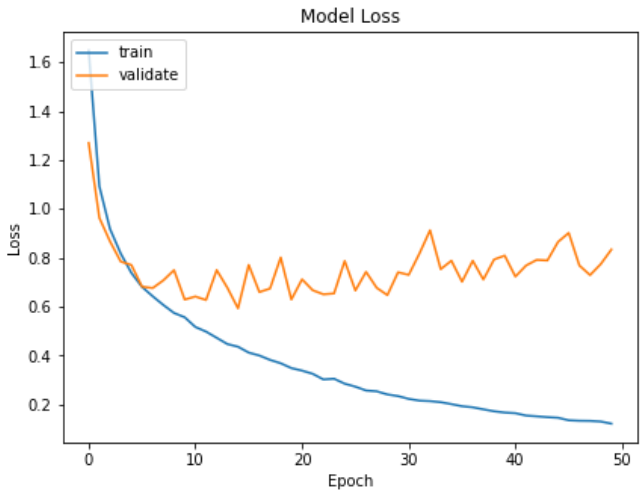
```
====================] - 22s 79ms/step - loss: 0.2725 - accuracy: 0.9066 - val_los

====================] - 22s 79ms/step - loss: 0.2568 - accuracy: 0.9124 - val_los

====================] - 22s 79ms/step - loss: 0.2535 - accuracy: 0.9141 - val_los

====================] - 21s 78ms/step - loss: 0.2407 - accuracy: 0.9179 - val_los
```

```
print(history_5.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plotmodelhistory(history_5)
```





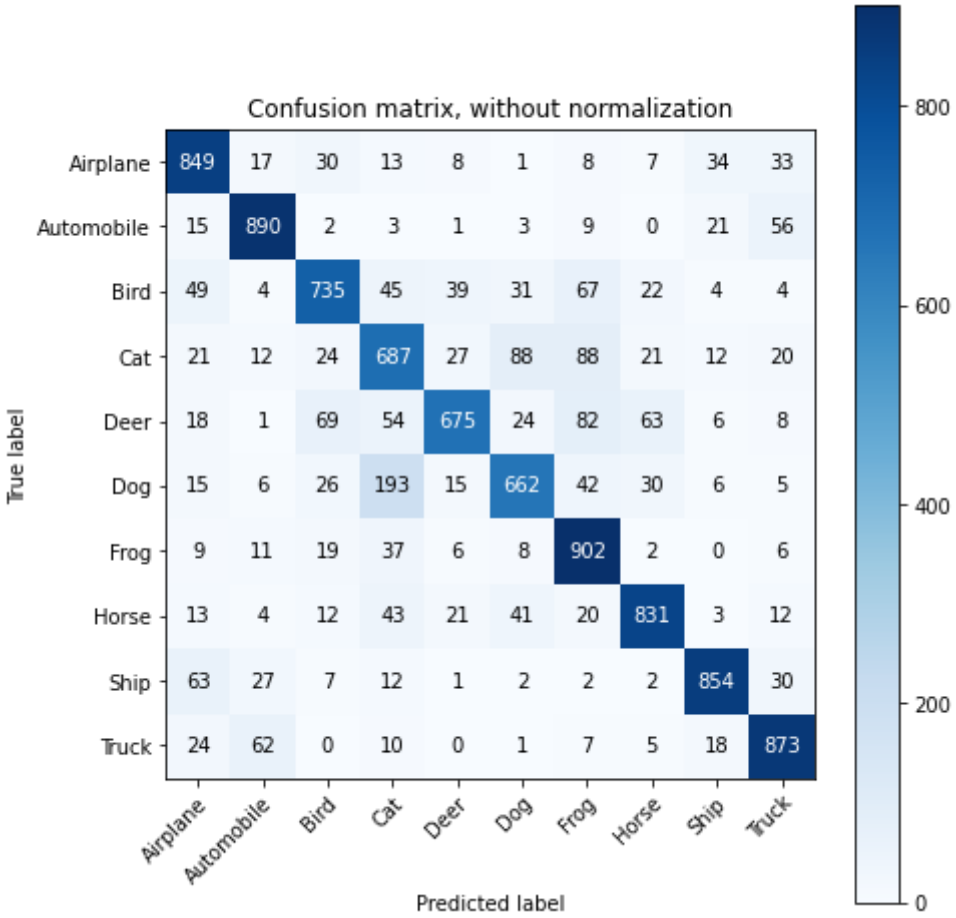

```
trained_model_5 = keras.models.load_model("best.keras")

# Score trained model.
scores = trained_model_5.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# make prediction.
pred = trained_model_5.predict(x_test)
```

```
313/313 [==============================] - 6s 15ms/step - loss: 0.6254 - accurac
Test loss: 0.62543123960495
Test accuracy: 0.795799970626831
```

```
predict_x=trained_model_5.predict(x_test)
y_pred=np.argmax(predict_x,axis=1)
y_true=np.argmax(y_test,axis=1)
```

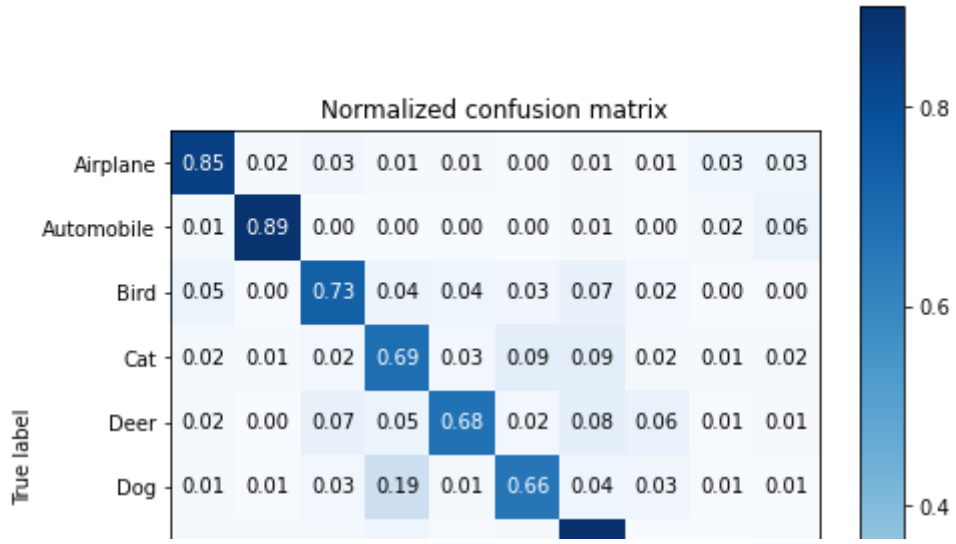
```
#Compute the confusion matrix
confusion_mtx=confusion_matrix(y_true,y_pred)

# Plot non-normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes=labels,
                      title='Confusion matrix, without normalization')
```

```
Confusion matrix, without normalization
<matplotlib.axes._subplots.AxesSubplot at 0x7facb5f77f90>
```




```
# Plot normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes=labels, normalize=True,
                      title='Normalized confusion matrix')
# plt.show()
```

```
Normalized confusion matrix
<matplotlib.axes._subplots.AxesSubplot at 0x7facb5dca690>
```



Summary table

Method	Test accuracy	
Baseline CNN	0.755	Bad at telling cats from
Transfer learning using pre-trained CNN (ResNet50) attempt #1	0.748	Disappointingly low, sigr
Transfer learning using (ResNet50) attempt #2	0.702	Disappointingly low, sigr
Transfer learning (ResNet50) with data augmentation and dropout (50 epochs) attempt #3	0.796	Best so far, signs of ove
		regularization increased

Question 1

Which classes are more likely to be misclassified by this model (Baseline CNN)? Does this make intuitive sense?

Classes that have similar structures and attributes are more likely to be misclassified by this model. For example, a dog is more likely to be misclassified as a cat, and a car is more likely to be misclassified as a truck. This is because dogs and cats have similar features and structures that the convolutional neural network “filters” for. A CNN works by learning local patterns that can be applied to any part of the image being analyzed by the network. In other words, the patterns are translation-invariant, and the CNN can recognize the learned feature or pattern in any part of the image. These patterns become more complex throughout the layers, as each layer uses the patterns learned in previous layers to learn increasingly complex and abstract features. Thus, shared features between cats and dogs, such as the shape of their ears, structure of their eyes, presence of whiskers, and structure of their fur, can activate similar filters when an image is passed through the network. This makes the network more likely to make a mistake and classify a dog as a cat or vise versa. The same principles apply to other classes that are more likely misclassified, such as a car being misclassified as a truck, or a bird being misclassified as an airplane.

Convolutional neural networks work in a very similar way to how images are interpreted in the visual world by humans, so the misclassifications make intuitive sense. Humans recognize images by using different attributes and features of the image and combining simple features to better classify what the image is. The model in this assignment works in similar ways, leading to mistakes that make intuitive sense. It makes sense that an airplane is misclassified as a bird, as both have wings and similar aerodynamic features that allow them to fly. In the same way some dogs look like cats and vise versa, as both are 4 legged furry domesticated pets with tails and whiskers. The mistakes made by the CNN misclassify images with similar features, which makes intuitive sense.

Question 2

Both attempts at transfer learning have resulted in rather disappointing results for validation/test accuracy. What seems to be the problem here?

Attempts one and two of transfer learning achieved relatively poor results with the best accuracy around 74%. Both models showed signs of overfitting and given enough epochs, both model's training accuracy seem to converge to around 100% accuracy. This indicates that the models have enough capacity to learn the relevant patterns in the dataset to perform the image classification well. However, the models need help increasing their generalization power. In the third attempt to apply transfer learning to the dataset, we applied regularization and data augmentation. These techniques were aimed at increasing the generalization power of the model by preventing the model from fitting too tightly to the training data. This worked because the test accuracy increased to about 80%, a 5-6% increase in accuracy over the previous two attempts. This indicates that the problem with the first two attempts at transfer learning was indeed the generalization power of the models. The transfer learning models in the first two attempts essentially took the convolutional base of the ResNet50 model and added dense layers. A better approach would be to try regularization, data augmentation, and fine-tuning to help the model adapt to the dataset better. For instance, another approach that could be experimented with is freezing the convolution base during training and only fine-tuning the top few convolution blocks of the model. This approach combined with the techniques used in the third attempt at transfer learning might achieve a test accuracy greater than 80%. Another technique that we applied to the third attempt that increased the test accuracy was allowing the model to train for a greater number of epochs and saving the model that achieved the highest validation accuracy. The saved model was then reloaded and evaluated on the test data. This approach allows us to save the model once overfitting becomes too prominent and reload the model with the best generalization and most power to identify relevant patterns in the data. Overall, the attempts at transfer learning mainly suffer from overfitting and a lack of generalization power.

Conclusions

In the first assignment we encountered the computer vision problem of classifying handwritten digits from the MNIST dataset, but we approached it from a non-convolutional neural network standpoint. However, this assignment allowed us to see how convolutional solutions can be applied to image classification problems, as convolutional neural networks are better suited for such computer vision tasks. The first part of this project focused on building and training a CNN from scratch to classify different categories of objects and animals, such as airplane, bird, cat, and automobile among others. The second part of the project focused on using pre-trained CNN base models for transfer learning. Both the from scratch and transfer learning solutions were similarly disappointing in terms of accuracy, achieving a score of roughly 0.75, and overfitting was present in both models. In our attempt to improve transfer learning, we ended with an accuracy of around 0.80, which was an improvement from the first two attempts. However, it was frustrating to get the improvement in accuracy. Training the model is a slow process (about 2 minutes), so every time that we experimented with a new parameter, we had to wait at least 2 minutes or more depending on the number of epochs we set. The first few times we experimented the accuracy went drastically down, and it took a dozen or so different modifications to finally get it to match the accuracy of attempt 2. The waiting time between each experiment, plus the dreaded decrease in accuracy after each training made this process frustrating.

This assignment furthered our familiarity with using CNNs to tackle computer vision problems. We understood the workflow of creating, training, and evaluating a CNN significantly better after finishing the assignment. The idea of transfer learning was also introduced to us in this assignment. This method is interesting because it allows for powerful results while reducing the development time and computational resources needed. It would be interesting to run more experiments with various pre-existing models and see how they could be applied to real world datasets and problems. Finally, this assignment further demonstrated that one of the main issues surrounding deep learning is the battle between overfitting and underfitting. Once the model is capable of overfitting to the training data, the aim is to achieve generalization and get the model to perform well on data it has never seen. As previously discussed, this was the most difficult part of developing the third attempt at transfer learning in this assignment.

✓ 1s completed at 8:04 PM