Machine Vision

CSE 489

Project 2

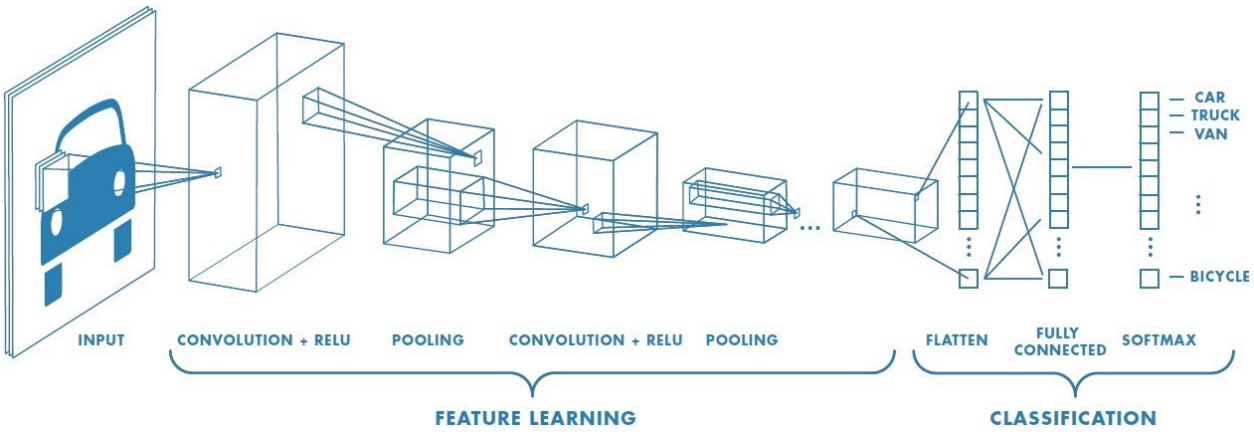
Submitted by:

Mohamed Abd El Hamed Moustafa - 16p8173
George Medhat Halim - 16p8197
Essam El-Din Amr - 16p9198

Date 12/1/2021
Senior 2
Mechatronics

Problem Definition:

The problem definition is that there are different types of objects in photos that need to be identified. The images can be of cars, cats, dogs, humans, planes ,etc. and it's required that an algorithm using Convolutional Neural Network with deep learning to classify those images and identify them with their correct label. The convolutional neural network consists of many layers usually it goes as follows, convolutional layer then max pooling layer then a flattening layer then a neural network layer then the output neural network layer. Most of those layers can be repeated many times to achieve a more complicated model to solve more complicated classification problems. Other layers can be added like dropout layers to reduce the effect of overfitting, batch normalization can also be added.



Importance:

Identifying objects using Convolutional Neural Networks s is a very important as it's one of the most well known way of classifications in the machine vision field. The classifications using Convolutional Neutral Networks has many applications like, Image recognition, Video analysis, Natural language processing, Anomaly detection, Drug discovery, Health risk assessment and biomarkers of aging discovery, Checkers game, Time series forecasting, and Cultural Heritage and 3D-datasets.

Methods and Algorithms:

Our own implemented classifier :

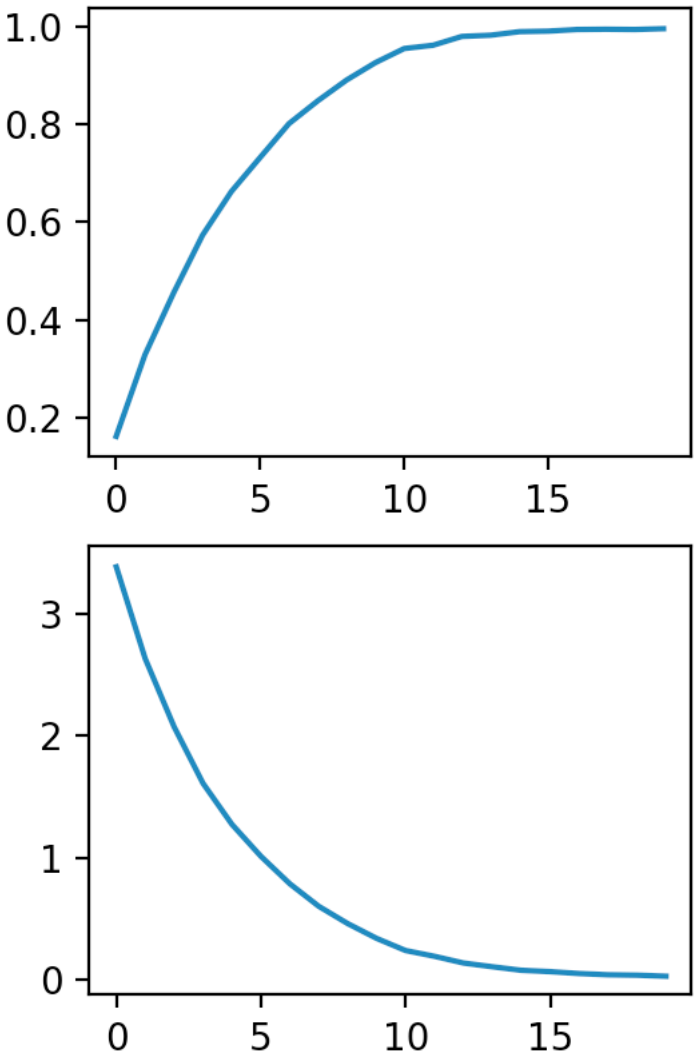
Firstly 50 class was chosen but the Catltech101 dataset has uneven images in each class so that would have trained the NN in an unbalanced way which would be unfair to some images and it would create false total accuracy as the classes with the very large training images would be learned perfectly and 15% of the total images would be taken for testing and most of them would have been classified correctly while the classes with lower number of images, the neural network would not train on them perfectly so their test accuracy would not be as good but due to their lower number it wouldn't affect the total accuracy as much so that would give false accuracy.

To over come this issue 50 images was chosen randomly for each class 36 images of them are used for training and 7 for testing and 7 for validation, 36 images for training is a very small number so mirror augmentation was used to increase the number of training images a bit. The images was also resized to be 300 by 300 pixels as not all images had the same dimensions and the has large number of pixels so the training was taking too long so it was reduced to 64 pixels by 64.

The images has been normalized and the training images has been shuffled, then a sequential model was created using 1 convolutional layer using kernel of size (3,3) and 64 filter followed by 1 max pooling layer followed by 1 convolutional layer using kernel of size (4,4) and 128 filter followed by 1 max pooling layer followed by 2 convolutional layer using kernel of size (5,5) and 246 filter followed by 1 max pooling layer followed by a batch normalization layer then a flatten layer then neural network layer of 1000 neuron followed by a drop out using 35% followed by the final output layer with 50 neurons.

The optimizer used is Adam and the loss used is the sparse categorical cross entropy and 20 epochs was used and it took about 15 minutes to train. The Training Accuracy was 99.89% and the Validation accuracy was 50% while the testing was 52%. Better could be achieved if i used all images in all classes as 2 or 3 classes have more than 400 image while most of them have only 50 images so when training on all images the couple 400 images classes will train very

nicely and 15 percent of those will result in great accuracy but it will falsely increase the accuracy as it's not fair as not all images trained using the same number of images and the classes that trained better have higher contribution in the total accuracy (due to have more images so 15% of those will be more than 15% of 50 images on the classes with less images) so we didn't opt to use this method and we used a fair 50 image per class as it's scientifically reasonable.



Layer (type)	Output Shape	Param #
Convolutional_layer1 (Conv2D	(None, 48, 48, 64)	1792
Maxpooling_2D1 (MaxPooling2D	(None, 24, 24, 64)	0
Convolutional_layer2 (Conv2D	(None, 21, 21, 128)	131200
Maxpooling_2D2 (MaxPooling2D	(None, 10, 10, 128)	0
Convolutional_layer3 (Conv2D	(None, 6, 6, 246)	787446
Convolutional_layer4 (Conv2D	(None, 2, 2, 246)	1513146
Maxpooling_2D3 (MaxPooling2D	(None, 1, 1, 246)	0
batch_normalization (BatchNo	(None, 1, 1, 246)	984
Flatten (Flatten)	(None, 246)	0
Hidden_layer (Dense)	(None, 1000)	247000
dropout (Dropout)	(None, 1000)	0
Output_layer (Dense)	(None, 50)	50050

Training accuracy and loss

Testing accuracy for each class using our own implemented

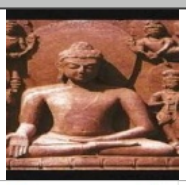
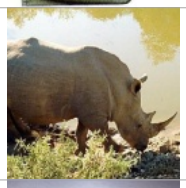
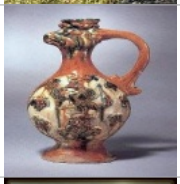
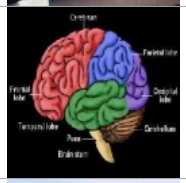
Image	Accuracy	Image	Accuracy
	57.14%		28.57%
	57.14%		42.85%
	42.85%		57.14%
	14.28%		14.28%
	100%		0%
	71.43%		42.85%
	71.43%		28.57%
	14.28%		57.14%

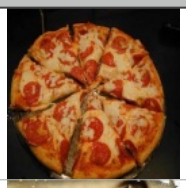
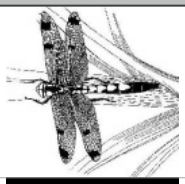
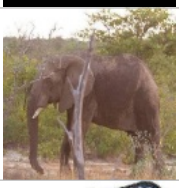
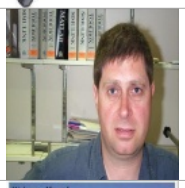
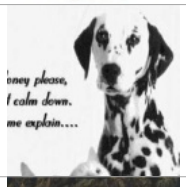
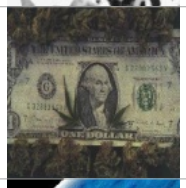
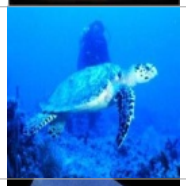
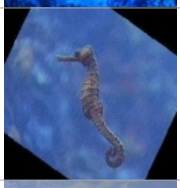
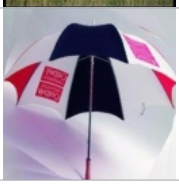
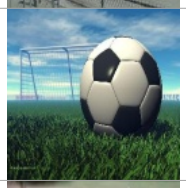
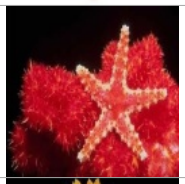
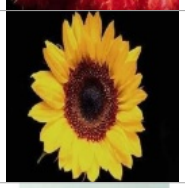
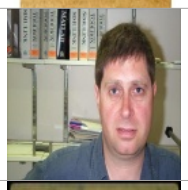
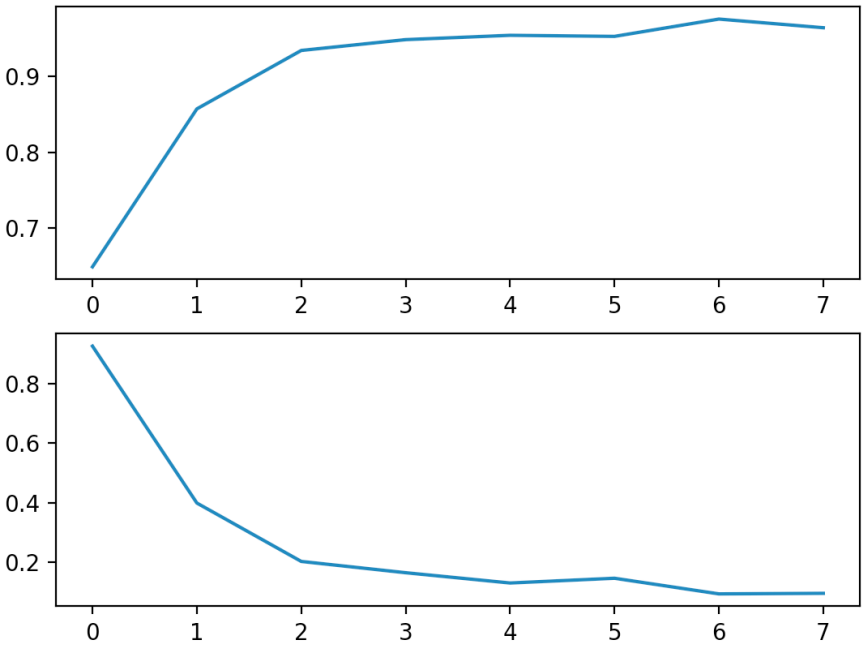
Image	Accuracy	Image	Accuracy
	71.43%		71.43%
	42.85%		0%
	28.57%		42.85%
	28.57%		28.57%
	28.57%		100%
	42.85%		85.71%
	71.42%		85.71%
	100%		28.57%
	28.57%		42.85%

Image	Accuracy	Image	Accuracy
	85.71%		57.14%
	71.42%		42.85%
	42.85%		57.14%
	57.14%		85.71%
	42.85%		85.71%
	57.14%		0%
	71.42%		100%
	42.85%		85.71%

I also tried using 200 images per category (only found 5 classes that had 200 images or more) on the model i created and the results were very promising as now it had 140 images to train on and 30 images for validation and another 30 for testing, The Training accuracy was 85% and the Testing accuracy was 82% which is very good so the short comings of the 50 class classifications is only due to the low number of images per class.

Image	Accuracy	Image	Accuracy
	100%		93.33%
	80%		96.67%
	40%		



Readymade ResNet50 classifier :

The ResNet50 model was implemented using 10 epochs. ResNet showed not very good results due to the limited number of training data and epochs as each epochs takes about 15 minutes so 10 epochs takes 2.5 hours which is very time consuming. This was done using 36 images for training ,7 for testing, and 7 for validation.

The training accuracy was 81.19% and the loss was 0.6777 while the validation accuracy was 37.43% and the loss was 3.1249 while the testing accuracy was 38.29 % and the loss was 2.9522.

Testing accuracy for each class using using ResNet50

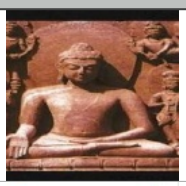
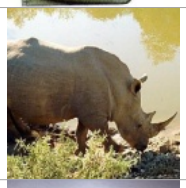
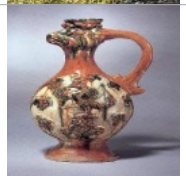
Image	Accuracy	Image	Accuracy
	42.85%		57.14%
	57.14%		42.85%
	42.85%		28.57%
	0%		42.85%
	85.71%		42.85%

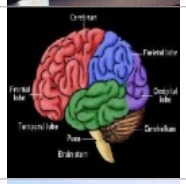
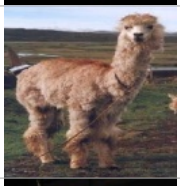
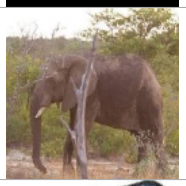
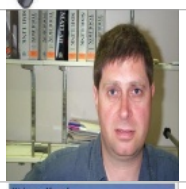
Image	Accuracy	Image	Accuracy
	14.28%		0%
	0%		0%
	28.57%		0%
	71.43%		57.14%
	14.28%		100%
	57.14%		14.28%
	57.14%		57.14%
	14.28%		28.57%
	14.28%		42.85%

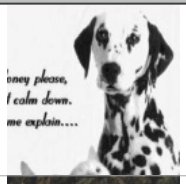
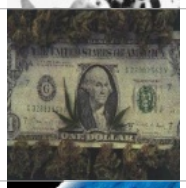
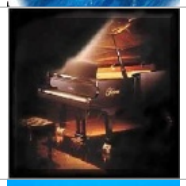
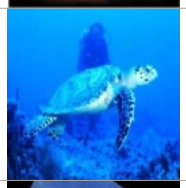
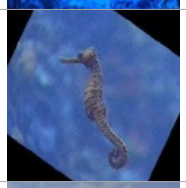
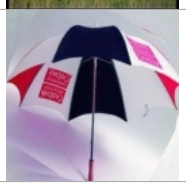
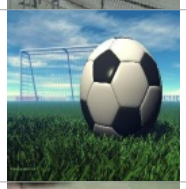
Image	Accuracy	Image	Accuracy
	42.85%		0%
	0%		28.57%
	57.14%		42.85%
	42.85%		42.85%
	28.57%		71.42%
	0%		42.85%
	0%		57.14%
	14.28%		42.85%
	14.28%		85.71%

Image	Accuracy	Image	Accuracy
	100%		100%
	0%		85.71%

Readymade Pre-Trained ResNet34 classifier :

Due to the not so good results of the previous ResNet50 model, i tried another readymade model but pre trained with the ImageNet weights using transfer learning. Wide range of augmentation was used in this algorithm including and not limited to cropping, rotating, and coloring. 5 epochs was used which took almost 2.5 hours. The results were very promising the training accuracy was 99.33% and the loss was 0.0029 , and the Validation accuracy was 98.38% and the loss was 0.0035 , and the Testing accuracy was 96.887%.

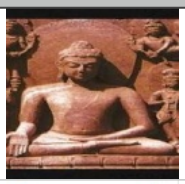
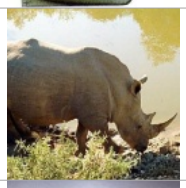
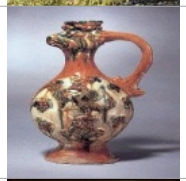
Image	Accuracy	Image	Accuracy
	100%		100%
	100%		100%
	100%		100%
	100%		100%
	100%		100%
	100%		100%

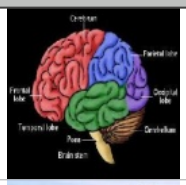
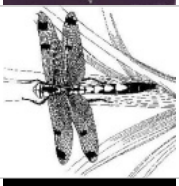
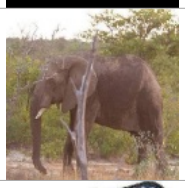
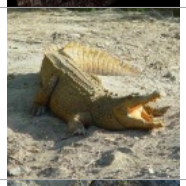
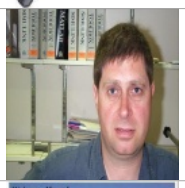
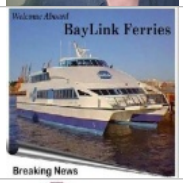
Image	Accuracy	Image	Accuracy
	100%		100%
	100%		100%
	84.21%		100%
	100%		100%
	100%		100%
	100%		100%
	100%		82.35%
	92.85%		100%
	40%		100%

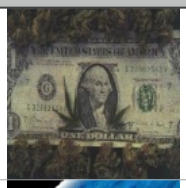
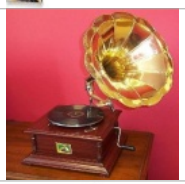
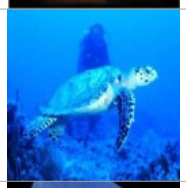
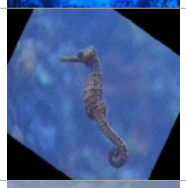
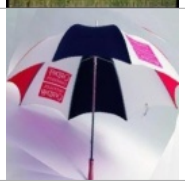
Image	Accuracy	Image	Accuracy
	100%		100%
	100%		100%
	93.75%		95.83%
	89.47%		87.5%
	100%		100%
	100%		100%
	100%		100%
	100%		100%
	100%		100%

Image	Accuracy	Image	Accuracy
	100%		84.21%

Conclusion :

Our own designed CNN model showed good results given the fact that it only trained on 36 images per class while the 5 classes with 200 images each showed great results (82% on the testing images) which shows that the problem was in the low number of training data not the design itself. The ResNet50 readymade model didn't show great results due to its complexity and time to train, the same number of epochs for our own designed CNN is about 8 times less than the time that took the ResNet50 to do the same number of epochs. While the pre trained ResNet34 using transfers learning showed as expected great results due to its pre-trained nature using the weights from ImageNet.

Appendix:

Our own model's code:

```
import numpy as np
import cv2
import random
import matplotlib.pyplot as plt
from tensorflow import keras as ks

epochs = 20
width = 50
height = 50
histogram = np.zeros([50])

Training_images = np.zeros([1800*2,width,height,3])
Training_labels = np.zeros([1800*2],dtype=int)
Validation_images = np.zeros([350,width,height,3])
Validation_labels = np.zeros([350],dtype=int)
Testing_images = np.zeros([350,width,height,3])
Testing_labels = np.zeros([350],dtype=int)

#index is a randomly generated list of 1800*2 number
#this list puts the images randomly in the training images array
index = list(range(0,1800*2))
random.shuffle(index)

indexVal =list(range(0,7*50))
random.shuffle(indexVal)

Train_counter = 0
Validation_counter = 0
Test_counter = 0
for i in range(50):
    for j in range(50):
        if j<36:
            if j <9:
                image1 = cv2.imread("Caltech101/" + str(i + 1) + "/image_000" +str(j + 1) + ".jpg")
                #print("Caltech101/" + str(i + 1) + "/image_000" + str(j + 1) + ".jpg")
            else:

                image1 = cv2.imread("Caltech101/" + str(i+1) + "/image_00" + str(j+1) + ".jpg")
                #print("Caltech101/" + str(i + 1) + "/image_00" + str(j + 1) + ".jpg")
            #resize the image to be exactly 300,300 pixels as not all images had the same shape
            image1 = image1.astype('float')
            resized_img1 = cv2.resize(image1, (width,height), interpolation = cv2.INTER_AREA)/255
            Training_images[[Train_counter], :, :, :] = resized_img1
            Training_labels[[Train_counter]] = i
            Train_counter=Train_counter+1

        if j>35 and j<43:
            image2 = cv2.imread("Caltech101/" + str(i + 1) + "/image_00" + str(j + 1) + ".jpg")
            image2 = image2.astype('float')
            resized_img2 = cv2.resize(image2, (width, height), interpolation=cv2.INTER_AREA)/255
            Validation_images[[Validation_counter],:,:,:] = resized_img2
            Validation_labels[[Validation_counter]] = i
            Validation_counter = Validation_counter + 1
        if j > 42 and j < 50:
            image3 = cv2.imread("Caltech101/" + str(i + 1) + "/image_00" + str(j + 1) + ".jpg")
            image3 = image3.astype('float')
            resized_img3 = cv2.resize(image3, (width, height), interpolation=cv2.INTER_AREA)/255
```

```
            Testing_images[Test_counter,:,:,:] = resized_img3
            Testing_labels[Test_counter] = i
            Test_counter = Test_counter + 1

for i in range(50):
    for j in range(36):


        image4 = cv2.imread("Caltech101/" + str(i + 1) + "/image_000" + str(j + 51) + ".jpg")
            #resize the image to be exactly 300,300 pixels as not all images had the same shape
        image4 = image4.astype('float')
        resized_img4 = cv2.resize(image4, (width, height), interpolation = cv2.INTER_AREA)/255
        Training_images[index[Train_counter], :, :, :] = resized_img4
        Training_labels[index[Train_counter]] = i
        Train_counter = Train_counter+1
print(Testing_labels)


cnn_model = ks.models.Sequential()
cnn_model.add(ks.layers.Conv2D(64,(3,3),activation='relu',input_shape = (width,height,3)
                               ,name='Convolutional_layer1'))
cnn_model.add(ks.layers.MaxPool2D((2,2),name = 'Maxpooling_2D1'))
cnn_model.add(ks.layers.Conv2D(128,(4,4),activation='relu',name='Convolutional_layer2'))
cnn_model.add(ks.layers.MaxPool2D((2,2),name = 'Maxpooling_2D2'))
cnn_model.add(ks.layers.Conv2D(246,(5,5),activation='relu',name='Convolutional_layer3'))
cnn_model.add(ks.layers.Conv2D(246,(5,5),activation='relu',name='Convolutional_layer4'))
cnn_model.add(ks.layers.MaxPool2D((2,2),name = 'Maxpooling_2D3'))
cnn_model.add(ks.layers.BatchNormalization())
cnn_model.add(ks.layers.Flatten(name='Flatten'))
cnn_model.add(ks.layers.Dense(1000,activation='relu',name='Hidden_layer'))
cnn_model.add(ks.layers.Dropout(0.35)) #0.35 was stable naw3an ma
cnn_model.add(ks.layers.Dense(50,activation='softmax',name='Output_layer'))
cnn_model.summary()

cnn_model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history = cnn_model.fit(Training_images,Training_labels,epochs=epochs,
                        validation_data=(Validation_images,Validation_labels) )


#Evaluate the model using the training data
training_loss, training_accuracy = cnn_model.evaluate(Training_images,Training_labels)
print('Training Accuracy {}' .format(round(float(training_accuracy),2)))

#Evaluate the model using the testing data
test_loss, test_accuracy = cnn_model.evaluate(Testing_images,Testing_labels)
print('Test Accuracy {}' .format(round(float(test_accuracy),2)))

#predictions = cnn_model.predict_classes(Testing_images)
predictions = np.argmax(cnn_model.predict(Testing_images), axis=-1)

Testing_accuracy = np.zeros([50],dtype=int)

for z in range (350):
    if Testing_labels[z] == predictions[z]:
        Testing_accuracy[Testing_labels[z]] =Testing_accuracy[Testing_labels[z]] +1
Testing_accuracy = np.divide(Testing_accuracy,7)
Testing_accuracy = np.multiply(Testing_accuracy,100)
print(predictions)
print(Testing_accuracy)


acc = history.history['accuracy']
loss = history.history['loss']

fig, ax = plt.subplots(nrows = 2)

ax[0].plot(acc, label="Training Accuracy")
ax[1].plot(loss, label="Training Loss")
plt.show()
```

ResNet50 Code:

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
import cv2
import numpy as np
from keras import layers
from keras.layers import Input, Add, Dense, Activation,
 ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D
from keras.models import Model, load_model
from keras.initializers import glorot_uniform
from keras.utils import plot_model
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
import keras.backend as K
import tensorflow as tf
import random
from tensorflow.python.framework import ops

ROWS = 64
COLS = 64
CHANNELS = 3
CLASSES = 50

EPOCHS = 10

width = 64
height = 64
histogram = np.zeros([50])
COLAB = False

Training_images = np.zeros([1800*2,width,height,3])
Training_labels = np.zeros([1800*2],dtype=int)
Validation_images = np.zeros([350,width,height,3])
Validation_labels = np.zeros([350],dtype=int)
Testing_images = np.zeros([350,width,height,3])
Testing_labels = np.zeros([350],dtype=int)

#index is a randomly generated list of 1800 number
#this list puts the images randomly in the training images array
index = list(range(0,1800*2))
#random.seed(4)
random.shuffle(index)

indexVal =list(range(0,7*50))
random.shuffle(indexVal)

def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return f"{h}:{m:>02}:{s:>05.2f}"

Train_counter = 0
Validation_counter = 0
Test_counter = 0
for i in range(50):
    for j in range(50):
        if j<36:
```

```
        if j<36:
            if j <9:
                image1 = cv2.imread("Caltech101/" + str(i + 1) + "/image_000" +str(j + 1) + ".jpg")
                #print("Caltech101/" + str(i + 1) + "/image_000" + str(j + 1) + ".jpg")
            else:

                image1 = cv2.imread("Caltech101/" + str(i+1) + "/image_00" + str(j+1) + ".jpg")
                #print("Caltech101/" + str(i + 1) + "/image_00" + str(j + 1) + ".jpg")
            #resize the image to be exactly 300,300 pixels as not all images had the same shape
            image1 = image1.astype('float')
            resized_img1 = cv2.resize(image1, (width,height), interpolation = cv2.INTER_AREA)/255
            Training_images[index[Train_counter], :, :, :] = resized_img1
            Training_labels[index[Train_counter]] = i
            Train_counter=Train_counter+1

        if j>35 and j<43:
            image2 = cv2.imread("Caltech101/" + str(i + 1) + "/image_00" + str(j + 1) + ".jpg")
            image2 = image2.astype('float')
            resized_img2 = cv2.resize(image2, (width, height), interpolation=cv2.INTER_AREA)/255
            Validation_images[indexVal[Validation_counter],:,:,:] = resized_img2
            Validation_labels[indexVal[Validation_counter]] = i
            Validation_counter = Validation_counter + 1
        if j > 42 and j < 50:
            image3 = cv2.imread("Caltech101/" + str(i + 1) + "/image_00" + str(j + 1) + ".jpg")
            image3 = image3.astype('float')
            resized_img3 = cv2.resize(image3, (width, height), interpolation=cv2.INTER_AREA)/255
            Testing_images[Test_counter ,:,:,:] = resized_img3
            Testing_labels[Test_counter ] = i
            Test_counter = Test_counter + 1
```

```
for i in range(50):
    for j in range(36):


        image4 = cv2.imread("Caltech101/" + str(i + 1) + "/image_000" + str(j + 51) + ".jpg")
            #resize the image to be exactly 300,300 pixels as not all images had the same shape
        image4 = image4.astype('float')
        resized_img4 = cv2.resize(image4, (width , height), interpolation = cv2.INTER_AREA)/255
        Training_images[index[Train_counter], :, :, :] = resized_img4
        Training_labels[index[Train_counter]] = i
        Train_counter = Train_counter+1

def convert_to_one_hot(Y, C):
    Y = np.eye(C)[Y.reshape(-1)].T
    return Y

Y_train = convert_to_one_hot(Training_labels, CLASSES).T
Y_test = convert_to_one_hot(Testing_labels, CLASSES).T
Y_validation = convert_to_one_hot(Validation_labels,CLASSES).T

X_train = Training_images
X_test = Testing_images
X_validation = Validation_images
```

```
print ("number of training examples =", X_train.shape[0])
print ("number of test examples =", X_test.shape[0])
print ("X_train shape:", X_train.shape)
print ("Y_train shape:", Y_train.shape)
print ("X_test shape:", X_test.shape)
print ("Y_test shape:", Y_test.shape)

def identity_block(X, f, filters, stage, block):
    # defining name basis
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    F1, F2, F3 = filters
    X_shortcut = X

    X = Conv2D(filters = F1, kernel_size = (1, 1), strides = (1,1),
               padding = 'valid', name = conv_name_base + '2a',
               kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1,1),
               padding = 'same', name = conv_name_base + '2b',
               kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1,1),
               padding = 'valid', name = conv_name_base + '2c',
               kernel_initializer = glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)
    return X

tf.compat.v1.reset_default_graph()

with tf.compat.v1.Session() as test:
    A_prev = tf.compat.v1.placeholder("float", [3, 4, 4, 6])
    X = np.random.randn(3, 4, 4, 6)
    A = identity_block(A_prev, f = 2, filters = [2, 4, 6], stage = 1, block = 'a')
    test.run(tf.compat.v1.global_variables_initializer())
    out = test.run([A], feed_dict={A_prev: X, K.learning_phase(): 0})
    print("out = ", out[0][1][1][0])

def convolutional_block(X, f, filters, stage, block, s=2):
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    F1, F2, F3 = filters
    X_shortcut = X

    ##### MAIN PATH #####
    X = Conv2D(F1, (1, 1), strides=(s, s), name=conv_name_base + '2a'
    , kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
    X = Activation('relu')(X)
```

```
169
170     X = Conv2D(filters=F2, kernel_size=(f, f), strides=(1, 1),
171                padding='same', name=conv_name_base + '2b',
172                kernel_initializer=glorot_uniform(seed=0))(X)
173     X = BatchNormalization(axis=3, name=bn_name_base + '2b')(X)
174     X = Activation('relu')(X)
175
176     X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1),
177                padding='valid', name=conv_name_base + '2c',
178                kernel_initializer=glorot_uniform(seed=0))(X)
179     X = BatchNormalization(axis=3, name=bn_name_base + '2c')(X)
180
181     ##### SHORTCUT PATH ####
182     X_shortcut = Conv2D(F3, (1, 1), strides=(s, s), name=conv_name_base + '1',
183                         kernel_initializer=glorot_uniform(seed=0))(X_shortcut)
184     X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)
185
186     # Final step: Add shortcut value to main path, and pass it through a RELU activation
187     X = Add()([X, X_shortcut])
188     X = Activation('relu')(X)
189
190     return X
191
192 tf.compat.v1.reset_default_graph()
193 with tf.compat.v1.Session() as test:
194     A_prev = tf.compat.v1.placeholder("float", [3, 4, 4, 6])
195     X = np.random.randn(3, 4, 4, 6)
196     A = convolutional_block(A_prev, f = 2, filters = [2, 4, 6], stage = 1, block = 'a')
197     test.run(tf.compat.v1.global_variables_initializer())
198     out = test.run([A], feed_dict={A_prev: X, K.learning_phase(): 0})
199     print("out = ",out[0][1][1][0])
200
201
202 def ResNet50(input_shape=(64, 64, 3), classes=2):
203     # Define the input as a tensor with shape input_shape
204     X_input = Input(input_shape)
205
206     # Zero-Padding
207     X = ZeroPadding2D((3, 3))(X_input)
208
209     # Stage 1
210     X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel_initializer=glorot_uniform(seed=0))(X)
211     X = BatchNormalization(axis=3, name='bn_conv1')(X)
212     X = Activation('relu')(X)
213     X = MaxPooling2D((3, 3), strides=(2, 2))(X)
214
215     # Stage 2
216     X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
217     X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
218     X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')
219
220     # Stage 3
221     X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
222     X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
223     X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
224     X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')
225
```

```
    X_shortcut = BatchNormalization(axis=3, name=bn_name_base + '1')(X_shortcut)

    # Final step: Add shortcut value to main path, and pass it through a RELU activation
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)

    return X

tf.compat.v1.reset_default_graph()
with tf.compat.v1.Session() as test:
    A_prev = tf.compat.v1.placeholder("float", [3, 4, 4, 6])
    X = np.random.randn(3, 4, 4, 6)
    A = convolutional_block(A_prev, f = 2, filters = [2, 4, 6], stage = 1, block = 'a')
    test.run(tf.compat.v1.global_variables_initializer())
    out = test.run([A], feed_dict={A_prev: X, K.learning_phase(): 0})
    print("out = ",out[0][1][1][0])


def ResNet50(input_shape=(64, 64, 3), classes=2):
    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)

    # Zero-Padding
    X = ZeroPadding2D((3, 3))(X_input)

    # Stage 1
    X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)

    # Stage 2
    X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')

    # Stage 3
    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    # Stage 4
    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    # Stage 5
    X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    # AVGPOOL.
    X = AveragePooling2D((2, 2), name='avg_pool')(X)

```

```
241
242     # output layer
243     X = Flatten()(X)
244     X = Dense(classes, activation='softmax', name='fc' + str(classes), kernel_initializer=glorot_uniform(seed=0))(X)
245     # Create model
246     model = Model(inputs=X_input, outputs=X, name='ResNet50')
247
248     return model
249
250 model = ResNet50(input_shape = (ROWS, COLS, CHANNELS), classes = CLASSES)
251 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
252 model.fit(X_train, Y_train, epochs = EPOCHS, batch_size = 64,validation_data=(X_validation,Y_validation))
253 Testing_accuracy = np.zeros([50],dtype=int)
254 preds = model.evaluate(X_test, Y_test)
255 print(preds)
256 print(Testing_accuracy)
257 predictions = np.argmax(model.predict(Testing_images), axis=-1)
258 for z in range (350):
259     if Y_test[z] == predictions[z]:
260         Testing_accuracy[Y_test[z]]  =Testing_accuracy[Y_test[z]]  +1
261 Testing_accuracy = np.divide(Testing_accuracy,7)
262 Testing_accuracy = np.multiply(Testing_accuracy,100)
263
264
265 print ("Loss = " + str(preds[0]))
266 print ("Test Accuracy = " + str(preds[1]))
267 model.summary()
268
```

ResNet34 Pre-Trained Code:

```python
import matplotlib.pyplot as plt
import matplotlib
import cv2
import os
import torch
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import time
import random
import pretrainedmodels
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from torchvision.transforms import transforms
from torch.utils.data import DataLoader, Dataset
from tqdm import tqdm

matplotlib.style.use('ggplot')
'''SEED Everything'''

def seed_everything(SEED=42):
    random.seed(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)
    torch.backends.cudnn.benchmark = True  # keep True if all the input have same size.
```

```python
SEED = 42
seed_everything(SEED=SEED)
'''SEED Everything'''

if torch.cuda.is_available():
    device = 'cuda'
else:
    device = 'cpu'

epochs = 5
BATCH_SIZE = 16

image_paths = list(paths.list_images('101_ObjectCategories'))
data = []
labels = []
label_names = []
for image_path in image_paths:
    label = image_path.split(os.path.sep)[-2]
    if label == 'BACKGROUND_Google':
        continue
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    data.append(image)
    label_names.append(label)
    labels.append(label)
data = np.array(data)
labels = np.array(labels)
print((labels[0]))

lb = LabelBinarizer()
labels = lb.fit_transform(labels)
print(f"Total number of classes: {len(lb.classes_)}")
```

```
count_arr = []
label_arr = []
for i in range(len(lb.classes_)):
    count = 0
    # print(lb.classes_[i])
    for j in range(len(label_names)):
        if lb.classes_[i] in label_names[j]:
            count += 1
    count_arr.append(count)
    label_arr.append(lb.classes_[i])

plt.figure(figsize=(18, 15))
plt.bar(label_arr, count_arr, )
plt.xticks(rotation='vertical')
#plt.show()

(X, x_val, Y, y_val) = train_test_split(data, labels,
                                        test_size=0.16,
                                        stratify=labels,
                                        random_state=42)
(x_train, x_test, y_train, y_test) = train_test_split(X, Y,
                                                      test_size=0.1903,
                                                      random_state=42)
print(f"x_train examples: {x_train.shape}\nx_test examples: {x_test.shape}\nx_val examples: {x_val.shape}")

train_transform = transforms.Compose(
    [transforms.ToPILImage(),
     transforms.Resize((224, 224)),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])])
val transform = transforms.Compose(
```

```
    [transforms.ToPILImage(),
     transforms.Resize((224, 224)),
     transforms.ToTensor(),
     transforms.Normalize(mean=[0.485, 0.456, 0.406],
                          std=[0.229, 0.224, 0.225])])


# custom dataset
class ImageDataset(Dataset):
    def __init__(self, images, labels=None, transforms=None):
        self.X = images
        self.y = labels
        self.transforms = transforms

    def __len__(self):
        return (len(self.X))

    def __getitem__(self, i):
        data = self.X[i][:]

        if self.transforms:
            data = self.transforms(data)

        if self.y is not None:
            return (data, self.y[i])
        else:
            return data


train_data = ImageDataset(x_train, y_train, train_transform)
val_data = ImageDataset(x_val, y_val, val_transform)
test_data = ImageDataset(x_test, y_test, val_transform)
```

```
# dataloaders
trainloader = DataLoader(train_data, batch_size=16, shuffle=True)
valloader = DataLoader(val_data, batch_size=16, shuffle=True)
testloader = DataLoader(test_data, batch_size=16, shuffle=False)


class ResNet34(nn.Module):
    def __init__(self, pretrained):
        super(ResNet34, self).__init__()
        if pretrained is True:
            self.model = pretrainedmodels.__dict__['resnet34'](pretrained='imagenet')
        else:
            self.model = pretrainedmodels.__dict__['resnet34'](pretrained=None)

        self.l0 = nn.Linear(512, len(lb.classes_))
        self.dropout = nn.Dropout2d(0.4)

    def forward(self, x):
        # get the batch size only, ignore (c, h, w)
        batch, _, _, _ = x.shape
        x = self.model.features(x)
        x = F.adaptive_avg_pool2d(x, 1).reshape(batch, -1)
        x = self.dropout(x)
        l0 = self.l0(x)
        return l0


model = ResNet34(pretrained=True).to(device)
print(model)

# optimizer
optimizer = optim.Adam(model.parameters(), lr=1e-4)
# loss function
```

```
criterion = nn.CrossEntropyLoss()

# validation function
def validate(model, dataloader):
    print('Validating')
    model.eval()
    val_running_loss = 0.0
    val_running_correct = 0
    with torch.no_grad():
        for i, data in tqdm(enumerate(dataloader), total=int(len(val_data) / dataloader.batch_size)):
            data, target = data[0].to(device), data[1].to(device)
            outputs = model(data)
            loss = criterion(outputs, torch.max(target, 1)[1])

            val_running_loss += loss.item()
            _, preds = torch.max(outputs.data, 1)
            val_running_correct += (preds == torch.max(target, 1)[1]).sum().item()

        val_loss = val_running_loss / len(dataloader.dataset)
        val_accuracy = 100. * val_running_correct / len(dataloader.dataset)
        print(f'Val Loss: {val_loss:.4f}, Val Acc: {val_accuracy:.2f}')

        return val_loss, val_accuracy

# training function
def fit(model, dataloader):
    print('Training')
    model.train()
    train_running_loss = 0.0
    train_running_correct = 0
    for i, data in tqdm(enumerate(dataloader), total=int(len(train_data) / dataloader.batch_size)):
        data, target = data[0].to(device), data[1].to(device)
```

```
            optimizer.zero_grad()
            outputs = model(data)
            # print(outputs)
            # print(torch.max(target, 1)[1])
            loss = criterion(outputs, torch.max(target, 1)[1])
            train_running_loss += loss.item()
            _, preds = torch.max(outputs.data, 1)
            train_running_correct += (preds == torch.max(target, 1)[1]).sum().item()
            loss.backward()
            optimizer.step()

        train_loss = train_running_loss / len(dataloader.dataset)
        train_accuracy = 100. * train_running_correct / len(dataloader.dataset)

        print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_accuracy:.2f}")

        return train_loss, train_accuracy


def class_acc_test(model, testloader):
    class_correct = list(0. for i in range(len(lb.classes_)))
    class_total = list(0. for i in range(len(lb.classes_)))
    with torch.no_grad():
        for data in testloader:
            inputs, target = data[0].to(device), data[1].to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            correct = (predicted == torch.max(target, 1)[1])
            for i in range(len(predicted)):
                label = torch.max(target, 1)[1][i]
                class_correct[label] += correct[i].item()
                class_total[label] += 1
    for i in range(len(lb.classes_)):
        print(f"Accuracy of {lb.classes_[i]}: {100 * (class_correct[i] / class_total[i]):.3f}")


train_loss, train_accuracy = [], []
val_loss, val_accuracy = [], []
print(f"Training on {len(train_data)} examples, validating on {len(val_data)} examples...")
start = time.time()
for epoch in range(epochs):
    print(f"Epoch {epoch + 1} of {epochs}")
    train_epoch_loss, train_epoch_accuracy = fit(model, trainloader)
    val_epoch_loss, val_epoch_accuracy = validate(model, valloader)
    train_loss.append(train_epoch_loss)
    train_accuracy.append(train_epoch_accuracy)
    val_loss.append(val_epoch_loss)
    val_accuracy.append(val_epoch_accuracy)
end = time.time()

print((end - start) / 60, 'minutes')

# torch.save(model.state_dict(), f"../outputs/models/resnet34_epochs{epochs}.pth")
# accuracy plots
plt.figure(figsize=(10, 7))
plt.plot(train_accuracy, color='green', label='train accuracy')
plt.plot(val_accuracy, color='blue', label='validataion accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('Accuracy.png')
plt.show()
```

```
# loss plots
plt.figure(figsize=(10, 7))
plt.plot(train_loss, color='orange', label='train loss')
plt.plot(val_loss, color='red', label='validataion loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('loss.png')
plt.show()

def test(model, dataloader):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            inputs, target = data[0].to(device), data[1].to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += target.size(0)
            correct += (predicted == torch.max(target, 1)[1]).sum().item()
    return correct, total


correct, total = test(model, testloader)
print('Accuracy of the network on test images: %0.3f %%' % (100 * correct / total))
class_acc_test(model, testloader)

```