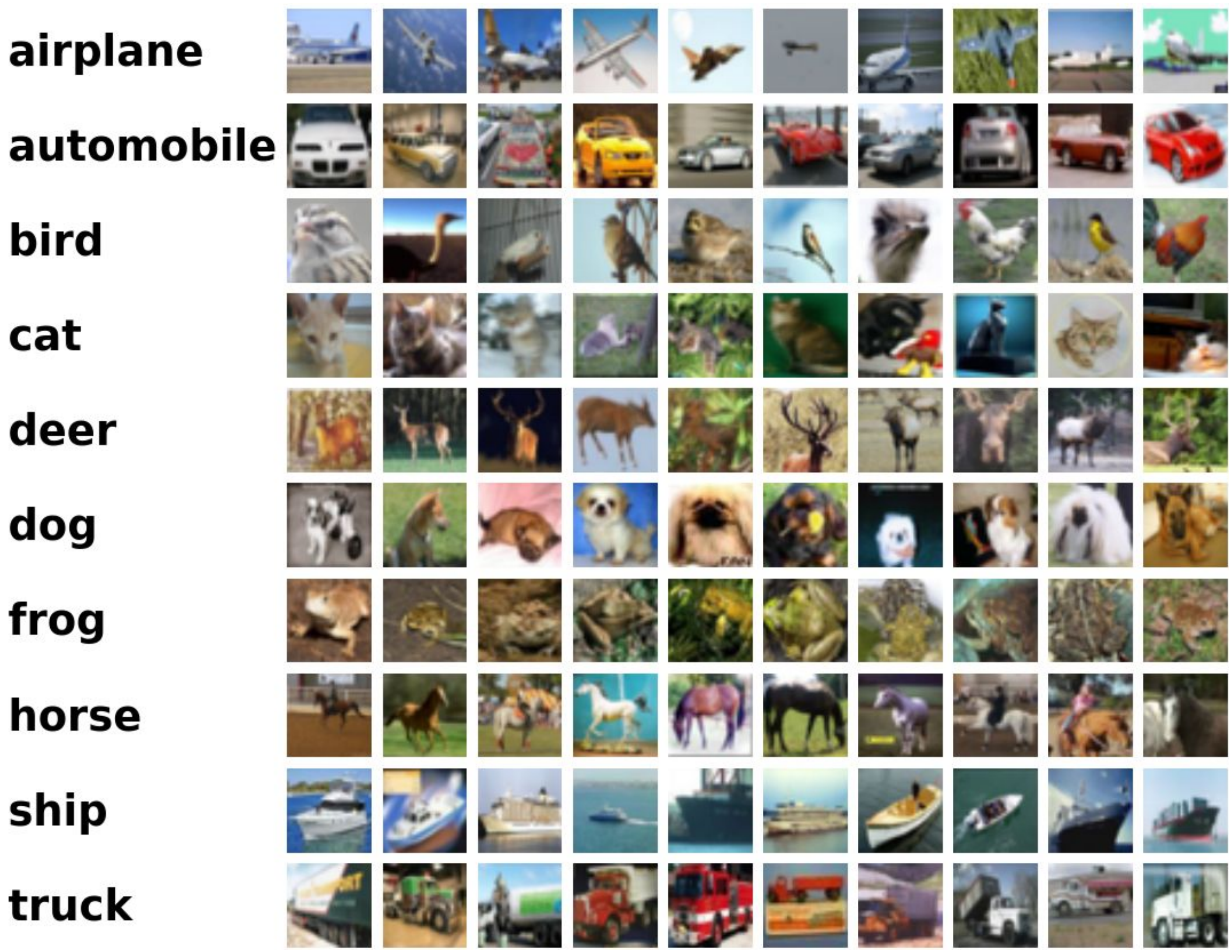
Machine Learning Assignment Week 8

Saul O'Driscoll (17333932)

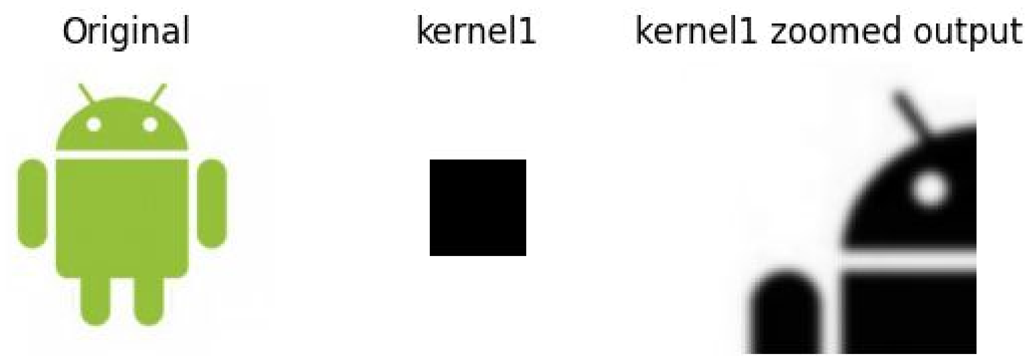
Dataset: *CIFAR-10*

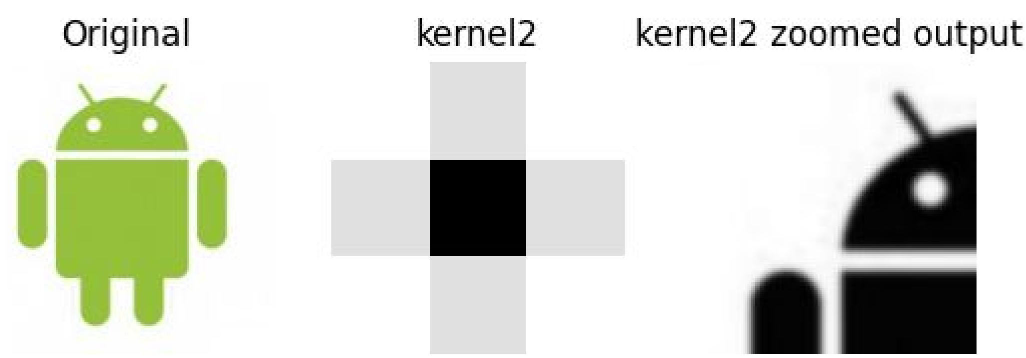


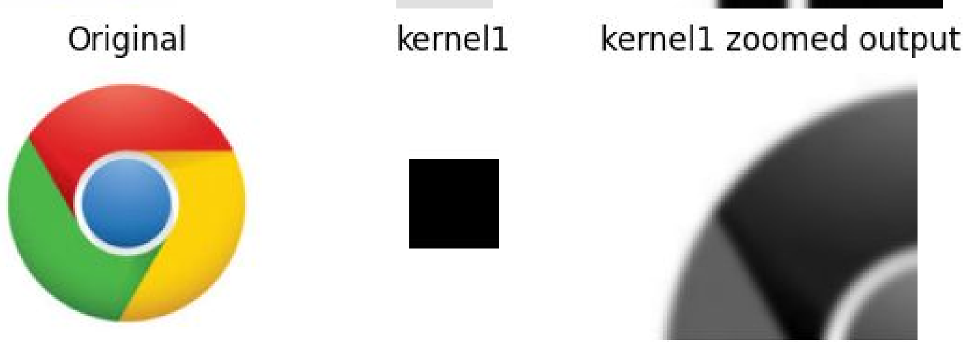
Saul O'Driscoll
Computer Science and Business

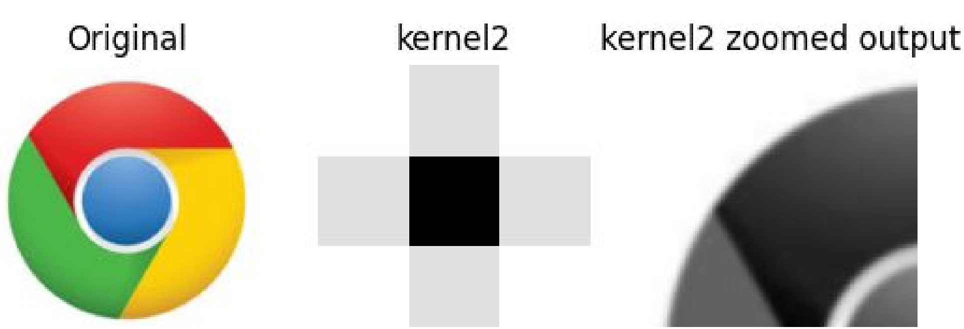
1.A + 1.B) CONVOLUTION

By combining vanilla python and numpy I was able to convolve images using a specified kernel. To increase the effect of the convolution I applied to convolution 6 times so one can see the effects of it better. I also zoomed the final image towards the top left quadrant to make the effects even more apparent. Below you can see the results of the code.





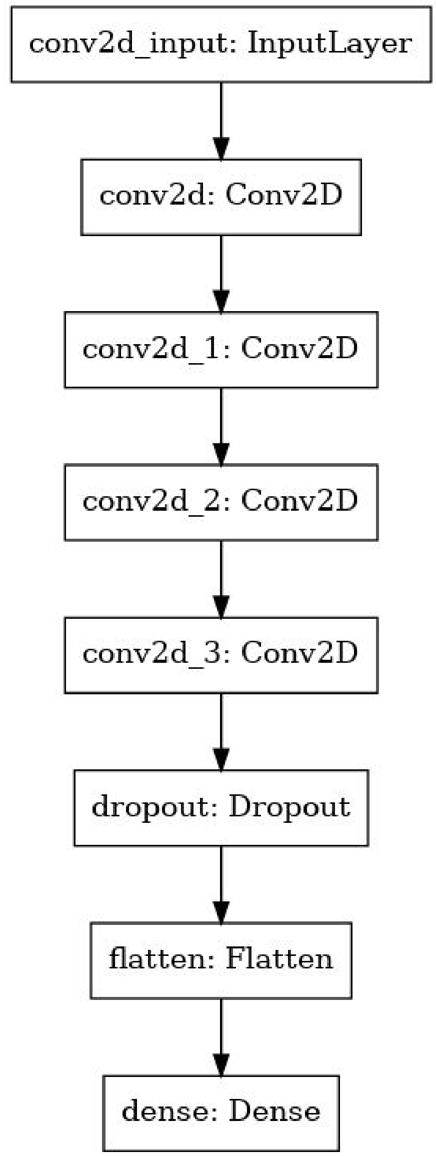




As you can see I did not opt to use a single channel from the RBG images but instead to convert them to grayscale before convolving the image because this is the more typical way of approaching Computer Vision tasks like this one.

2.A)

To the right we have a picture of our network.



Each layer has an input size and an output size. These can be the same or different depending on what each layer does.

There are 4 Convolutional layers and 1 Dropout layer.

- Conv2D #0
    - Input Size: 32x32
    - Output Size: 16x16
    - Kernel Size: 3x3
- Conv2D #1
    - Input Size: 16x16
    - Output Size: 16x16
    - Strides: 2x2
    - Kernel Size: 3x3
- Conv2D #2
    - Input Size: 16x16
    - Output Size: 32x32
    - Kernel Size: 3x3
- Conv2D #3
    - Input Size: 32x32
    - Output Size: 8x8
    - Strides: 2x2
    - Kernel Size: 3x3

B 1)

How many parameters does keras say this model has?

- Total params: 37,146

Which layer has the most parameters, and why?

- The output layer has the most because it essentially amounts to a fully connected layer as it is flattened. This means every neuron is connected to every other neuron which means there is the most amount of learned parameters here.

How does the performance on the test data compare with the performance on the training data?

The accuracy on the training data is higher as was expected. It came in at 0.61 ~ 61%. The accuracy on the test data was 0.48 ~ 48%

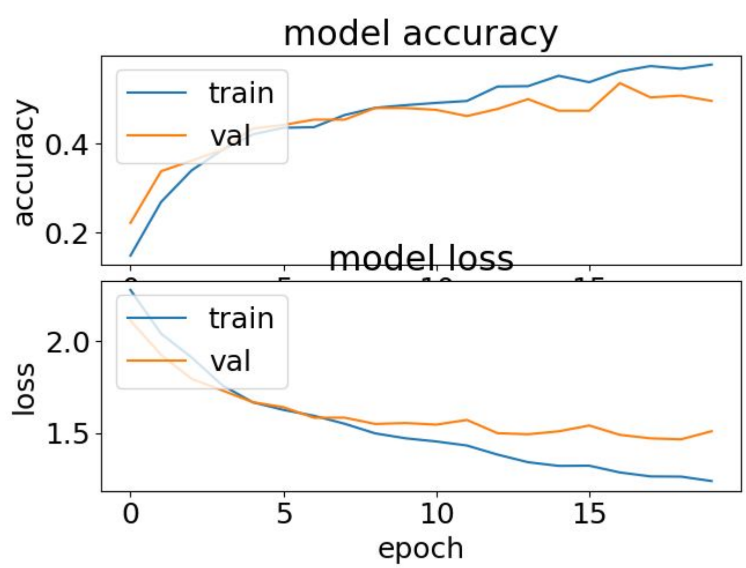
Compare this performance against a simple baseline e.g. always predicting the most common label.

There are 10 different classes and they are all equally common in the dataset. Given this our baseline predictor for a single class would be accurate 10% of the time on a random sample of data.

B 2)

What diagnostics, if any, about over/under-fitting can you deduce from this plot?

We can see that the model if overfit when val goes above train and underfit when the val line trends below train.



B 3)
How does the prediction accuracy on the training and test data vary with the amount of training data?

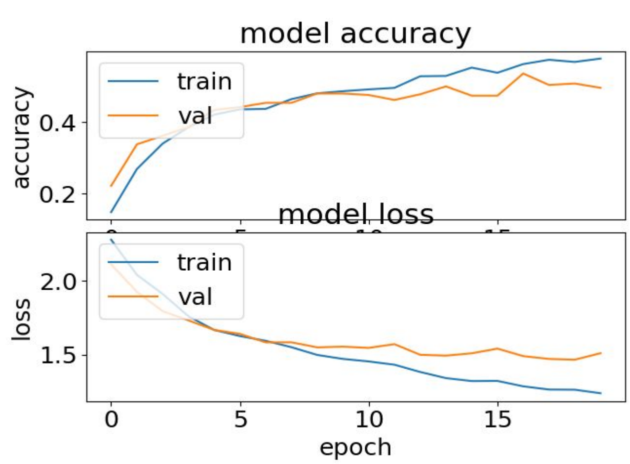
Accuracy for the test data.

5k training samples = 0.49 - 49% accuracy
10k training samples = 0.56 - 56% accuracy
20k training samples = 0.63 ~ 63% accuracy
40k training samples = 0.69 ~ 69% accuracy

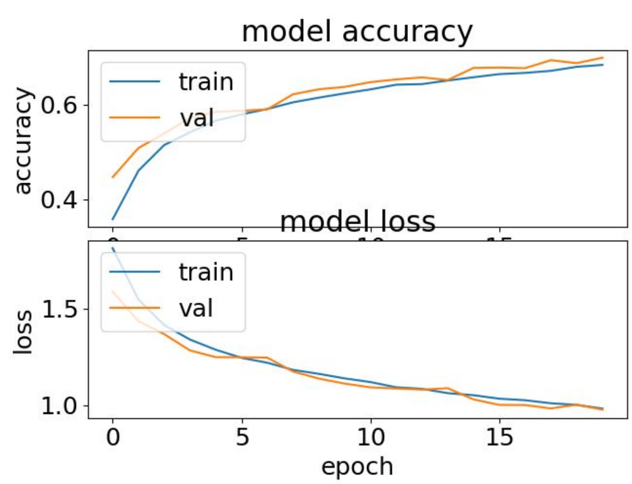
The accuracy increases with the amount of training samples that the network is shown.

Look at the plot of the "history" variable for each run, how does it vary with the amount of training data used?

5K examples



40k examples



Time (measured using the time library):
31s - 5000 training examples
69s - 10000 training examples
128s - 20000 training examples
261s - 40000 training examples

B4)

Here are values of different L1 values.

Time:31 - L_1: 100 => 0.10% accuracy
Time:31 - L_1: 10 => 0.10% accuracy
Time:31 - L_1: 1 => 0.10% accuracy
Time:32 - L_1: 0 => 0.50% accuracy
Time:31 - L_1: 0.1 => 0.32% accuracy
Time:30 - L_1: 0.0001 => 0.49% accuracy

C1+2)

When adding the MaxPooling layers we get a faster training time and higher accuracy for 5000 examples.

Total params: 25,578
Time: 24s ***vs.*** 31s using previous layers
Accuracy: 0.51 ~ 51% on test data ***vs.*** 0.49 - 49% accuracy using previous layers

Code for Part A:

```
import matplotlib.pyplot as plt
import numpy as np
import math
from PIL import Image

def rgb2gray(rgb):
   return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

def apply_kernel(image, kernel):
   m, n = kernel.shape
   if (m == n):
       y, x = image.shape
       y = y - m + 1
       x = x - m + 1
       new_image = np.zeros((y,x))
       for i in range(y):
           for j in range(x):
               new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)

   return new_image

def convo(image, kern):
   imgOut = image
   if (kern.shape[0]==kern.shape[1]):
       imgOut = apply_kernel(image,kern)
   return imgOut

test_files = ["android.jpg", "index.jpeg"]

for pic in test_files:
   im = np.asarray(Image.open(pic))
   gray = rgb2gray(im)

   kernel1 = np.array([[-1, -1, -1], [-1, -8, -1], [-1, -1, -1]])
   kernel2 = np.array([[0, -1, 0], [-1, -8, -1], [0, -1, 0]])

   print("Convolution 1")
   kernel1_result = convo(gray,kernel1)
```

```
    kernel2_result = convo(gray,kernel2)

    i = 2
    while(i <= 6):
        print("Convolution {0}".format(i))
        kernel1_result = convo(kernel1_result,kernel1)
        kernel2_result = convo(kernel2_result,kernel2)
        i += 1

    f, axarr = plt.subplots(2,3)

    k1y, k1x = kernel1_result.shape
    k2y, k2x = kernel2_result.shape

    zoom = 2.2

    axarr[0,0].set_title("Original")
    axarr[0,0].axis('off')
    axarr[0,0].imshow(im)
    axarr[1,0].set_title("Original")
    axarr[1,0].axis('off')
    axarr[1,0].imshow(im)

    axarr[0,1].set_title("kernel1")
    axarr[0,1].axis('off')
    axarr[0,1].imshow(kernel1, cmap="gray")
    axarr[1,1].set_title("kernel2")
    axarr[1,1].axis('off')
    axarr[1,1].imshow(kernel2, cmap="gray")

    axarr[0,2].set_title("kernel1 zoomed output")
    axarr[0,2].axis('off')

axarr[0,2].imshow(kernel1_result[:math.floor(k1x/zoom),:math.floor(k1y/zoo
m)], cmap="gray")
    axarr[1,2].set_title("kernel2 zoomed output")
    axarr[1,2].axis('off')

axarr[1,2].imshow(kernel2_result[:math.floor(k2x/zoom),:math.floor(k2y/zoo
m)], cmap="gray")
```

```
plt.show()
plt.close()
```

Code for Part B + C:

```
import pickle
import time
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from keras.layers import Dense, Dropout, Activation, Flatten,
BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, LeakyReLU
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
import sys

# Model / data parameters
num_classes = 10
input_shape = (32, 32, 3)

   # the data, split between train and test sets
l1_arr = [0.0001,0.1,0,1,10,100]
for l in l1_arr:
   (x_train, y_train), (x_test, y_test) =
keras.datasets.cifar10.load_data()
   n=5000
   x_train = x_train[1:n]; y_train=y_train[1:n]
   #x_test=x_test[1:500]; y_test=y_test[1:500]

   # Scale images to the [0, 1] range
   x_train = x_train.astype("float32") / 255
   x_test = x_test.astype("float32") / 255
```

```
    print("orig x_train shape:", x_train.shape)

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    model = "normal"
    if model=="saved":
        model = keras.models.load_model("cifar.model")
    elif(model=="maxpool"):
        model = keras.Sequential()
        model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
        model.add(MaxPooling2D(pool_size=2, strides=2, padding='same'))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        model.add(MaxPooling2D(pool_size=2, strides=2, padding='same'))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(0.0001)))
        model.compile(loss="categorical_crossentropy", optimizer='adam',
metrics=["accuracy"])
        model.summary()

        batch_size = 128
        epochs = 20
        start = time.time()
        history = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.1)
        end = time.time()
        model.save("cifar.model")
        plt.subplot(211)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('model accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        plt.subplot(212)
        plt.plot(history.history['loss'])
```

```
        plt.plot(history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss'); plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        # print("Time:" + str(end - start) + " L1: " + str(l))
        plt.show()
    else:
        model = keras.Sequential()
        model.add(Conv2D(16, (3,3), padding='same',
input_shape=x_train.shape[1:],activation='relu'))
        model.add(Conv2D(16, (3,3), strides=(2,2), padding='same',
activation='relu'))
        model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
        model.add(Conv2D(32, (3,3), strides=(2,2), padding='same',
activation='relu'))
        model.add(Dropout(0.5))
        model.add(Flatten())
        model.add(Dense(num_classes,
activation='softmax',kernel_regularizer=regularizers.l1(l)))
        model.compile(loss="categorical_crossentropy", optimizer='adam',
metrics=["accuracy"])
        model.summary()

        batch_size = 128
        epochs = 20
        start = time.time()
        history = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=0.1)
        end = time.time()
        model.save("cifar.model")
        plt.subplot(211)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('model accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'val'], loc='upper left')
        plt.subplot(212)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
```

```
    plt.title('model loss')
    plt.ylabel('loss'); plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    print("Time:" + str(end - start) + " L1: " + str(l))
    plt.show()


preds = model.predict(x_train)
y_pred = np.argmax(preds, axis=1)
y_train1 = np.argmax(y_train, axis=1)
print(classification_report(y_train1, y_pred))
print(confusion_matrix(y_train1,y_pred))

preds = model.predict(x_test)
y_pred = np.argmax(preds, axis=1)
y_test1 = np.argmax(y_test, axis=1)
print(classification_report(y_test1, y_pred))
print(confusion_matrix(y_test1,y_pred))
```