IMAGE CLASSIFICATION

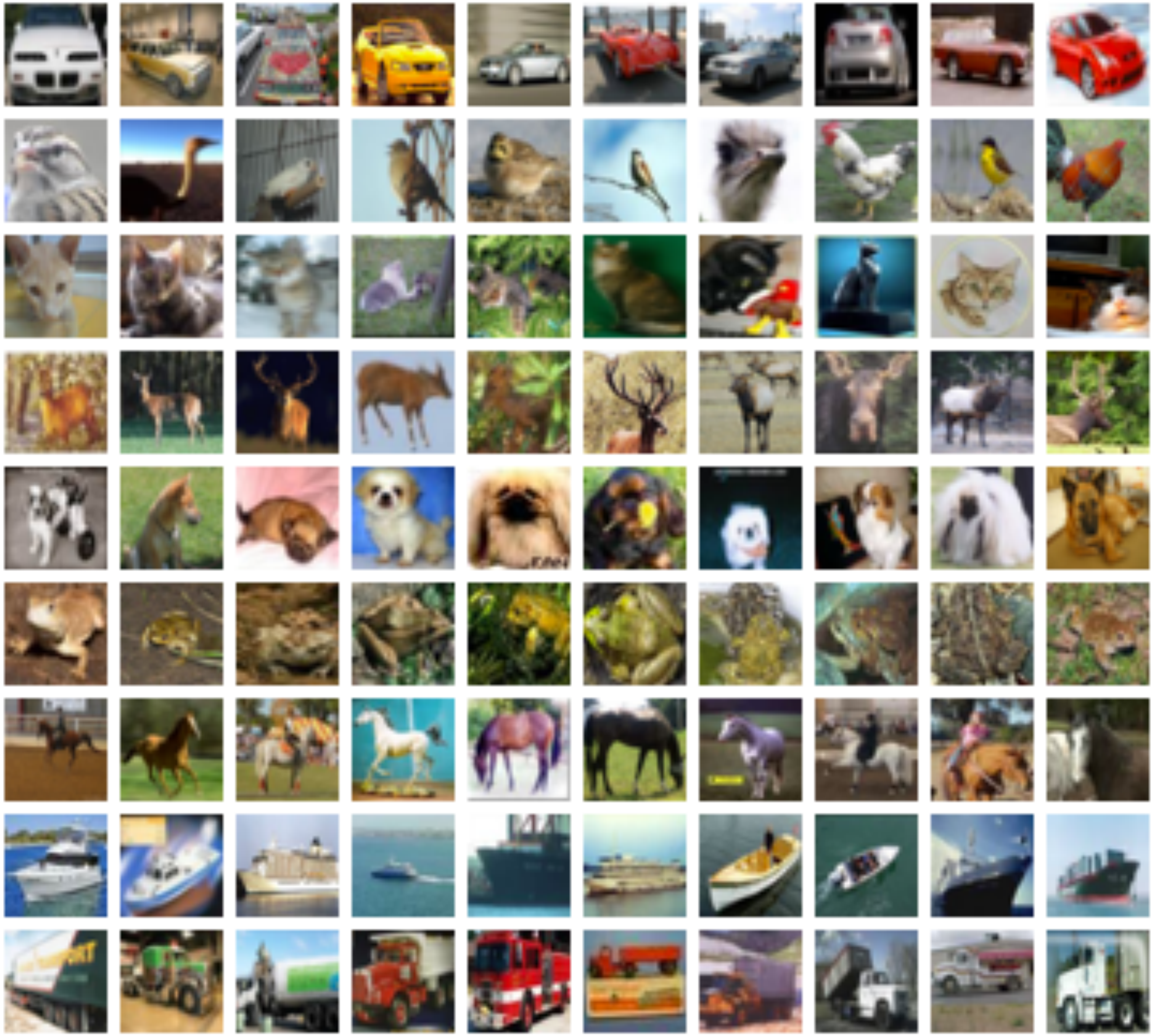
SUBMITTED BY:

Team 3B

Seerat Chhabra, Mahindra Venkat Lukka, Chris Chou, Yin Chen Wang

ABOUT DATA

- CIFAR -10 dataset (Canadian Institute For Advanced Research)
- Commonly used data for machine learning algorithms
- Contains 60,000 , 32*32 color images for 10 classes
- Classes are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks



CIFAR -10 DATASET

STEPS

1. Train & Test data from CIFAR-10
2. Convert Target variable to binary class matrix
3. Normalize pixel data
4. Train and Validation data out of Train data
5. Create CNN model by adding convolutional layer, maxpooling , feed forward neural network

TRAIN & TEST DATA FROM CIFAR-10

```
[2] # The data, split between train and test sets:
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    print('x_train shape:', x_train.shape)
    print('y_train shape:', y_train.shape)
    print('x_test shape:', x_test.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
x_test shape: (10000, 32, 32, 3)
50000 train samples
10000 test samples
```

Training data – 50,000 images of size 32*32*3
Test data – 10,000 images of size 32*32*3

```
[ ] y_train
```

```
array([[6],
       [9],
       [9],
       ...,
       [9],
       [1],
       [1]], dtype=uint8)
```

Target variable with 1-10

CONVERT TARGET VARIABLE TO BINARY CLASS MATRIX

```
# Convert class target variable to binary class matrices.
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print(y_train)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]]
```

NORMALIZE PIXEL DATA

```
#convert to float
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
#normalise data by dividing by pixel max range value of 255
x_train /= 255
x_test /= 255
```

Before normalize

```
x_train
```

```
array([[[[ 59,  62,  63],
         [ 43,  46,  45],
         [ 50,  48,  43],
         ...,
         [158, 132, 108],
         [152, 125, 102],
         [148, 124, 103]],

        [[ 16,  20,  20],
         [  0,   0,   0],
         [ 18,   8,   0],
         ...,
         [123,  88,  55],
         [119,  83,  50],
         [122,  87,  57]],

        [[ 25,  24,  21],
         [ 16,   7,   0],
         [ 49,  27,   8],
         ...,
         [118,  84,  50],
         [120,  84,  50],
         [109,  73,  42]],

        ...,

        [[208, 170,  96],
         [201, 153,  34],
         [198, 161,  26],
```

After normalize

```
x_train
```

```
array([[[[0.23137255, 0.24313726, 0.24705882],
         [0.16862746, 0.18039216, 0.1764706 ],
         [0.19607843, 0.1882353 , 0.16862746],
         ...,
         [0.61960787, 0.5176471 , 0.42352942],
         [0.59607846, 0.49019608, 0.4       ],
         [0.5803922 , 0.4862745 , 0.40392157]],

        [[0.0627451 , 0.07843138, 0.07843138],
         [0.        , 0.        , 0.        ],
         [0.07058824, 0.03137255, 0.        ],
         ...,
         [0.48235294, 0.34509805, 0.21568628],
         [0.46666667, 0.3254902 , 0.19607843],
         [0.47843137, 0.34117648, 0.22352941]],

        [[0.09803922, 0.09411765, 0.08235294],
         [0.0627451 , 0.02745098, 0.        ],
         [0.19215687, 0.10588235, 0.03137255],
         ...,
         [0.4627451 , 0.32941177, 0.19607843],
         [0.47058824, 0.32941177, 0.19607843],
         [0.42745098, 0.28627452, 0.16470589]],

        ...,

        [[0.8156863 , 0.6666667 , 0.3764706 ],
         [0.7882353 , 0.6       , 0.13333334]
```

VALIDATION DATA SET

```
#split validation set
from sklearn.model_selection import train_test_split
x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size=0.2,random_state=0)
```

80-20 spit for Train and Validation data set

1

CNN MODEL

```
from keras.models import *
from keras.layers import *
def build_model(out_dims, img_size):
    inputs_dim = Input((img_size, img_size, 3))

    x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation = 'relu')(inputs_dim)
    x = Conv2D(64, (3, 3), strides=(1, 1), padding='same',activation = 'relu')(x)
    x = MaxPool2D(pool_size=(2, 2))(x)

    x = Conv2D(128, (3, 3), strides=(1, 1), padding='same',activation = 'relu')(x)
    x = Conv2D(128, (3, 3), strides=(1, 1), padding='same',activation = 'relu')(x)
    x = MaxPool2D(pool_size=(2, 2))(x)

    x = Conv2D(256, (3, 3), strides=(1, 1), padding='same',activation = 'relu')(x)
    x = Conv2D(256, (3, 3), strides=(1, 1), padding='same',activation = 'relu')(x)
    x = MaxPool2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x_flat = Flatten()(x)
    dp_1 = Dropout(0.4)(x_flat)
    fc2 = Dense(out_dims)(dp_1)
    fc2 = Activation('softmax')(fc2)

    build_model = Model(inputs=inputs_dim, outputs=fc2)
    return model

model_2 = build_model(out_dims=10, img_size=32)
```

6 convolutional layer

1. 2 - 64 filters with size (3,3)
2. 2- 128 filters with size (3,3)
3. 2- 256 filters with size (3,3)
   a) Random dropout of nodes at 25% rate

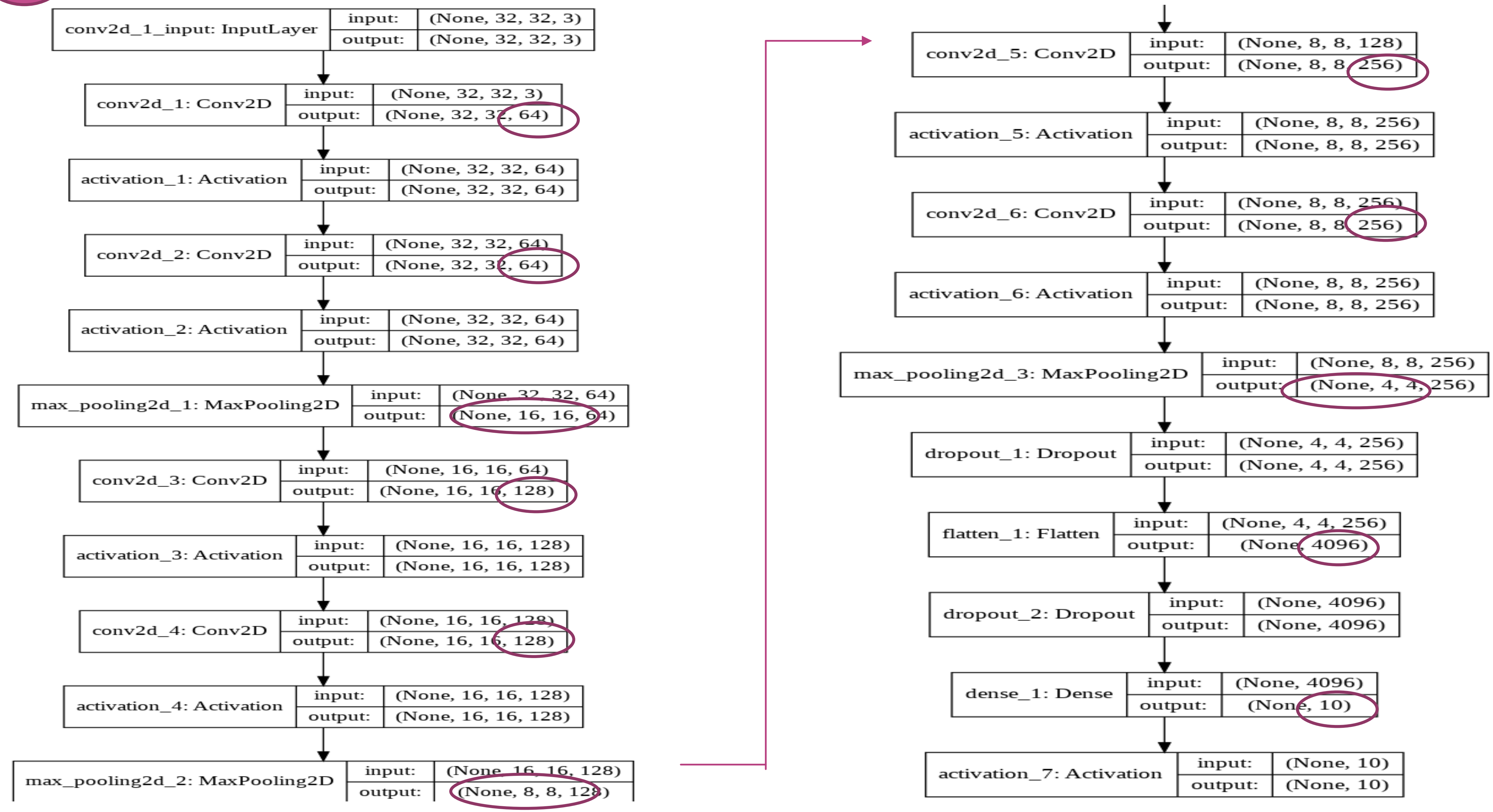
Flattening

Random dropout of nodes at 40% rate

Softmax layer with output 10 classifiers

2

CNN MODEL


conv2d_1_input: InputLayer
input: (None, 32, 32, 3)
output: (None, 32, 32, 3)
conv2d_1: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 64)
activation_1: Activation
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
conv2d_2: Conv2D
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
activation_2: Activation
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 32, 32, 64)
output: (None, 16, 16, 64)
conv2d_3: Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 128)
activation_3: Activation
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
conv2d_4: Conv2D
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
activation_4: Activation
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 16, 16, 128)
output: (None, 8, 8, 128)
conv2d_5: Conv2D
input: (None, 8, 8, 128)
output: (None, 8, 8, 256)
activation_5: Activation
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
conv2d_6: Conv2D
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
activation_6: Activation
input: (None, 8, 8, 256)
output: (None, 8, 8, 256)
max_pooling2d_3: MaxPooling2D
input: (None, 8, 8, 256)
output: (None, 4, 4, 256)
dropout_1: Dropout
input: (None, 4, 4, 256)
output: (None, 4, 4, 256)
flatten_1: Flatten
input: (None, 4, 4, 256)
output: (None, 4096)
dropout_2: Dropout
input: (None, 4096)
output: (None, 4096)
dense_1: Dense
input: (None, 4096)
output: (None, 10)
activation_7: Activation
input: (None, 10)
output: (None, 10)

CNN MODEL

```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(width_shift_range = 0.1,
                             height_shift_range = 0.1,
                             horizontal_flip = True,
                             zoom_range = 0.3)


batch_size = 64
train_generator = datagen.flow(x_train,y_train,batch_size=batch_size, shuffle=False)
```

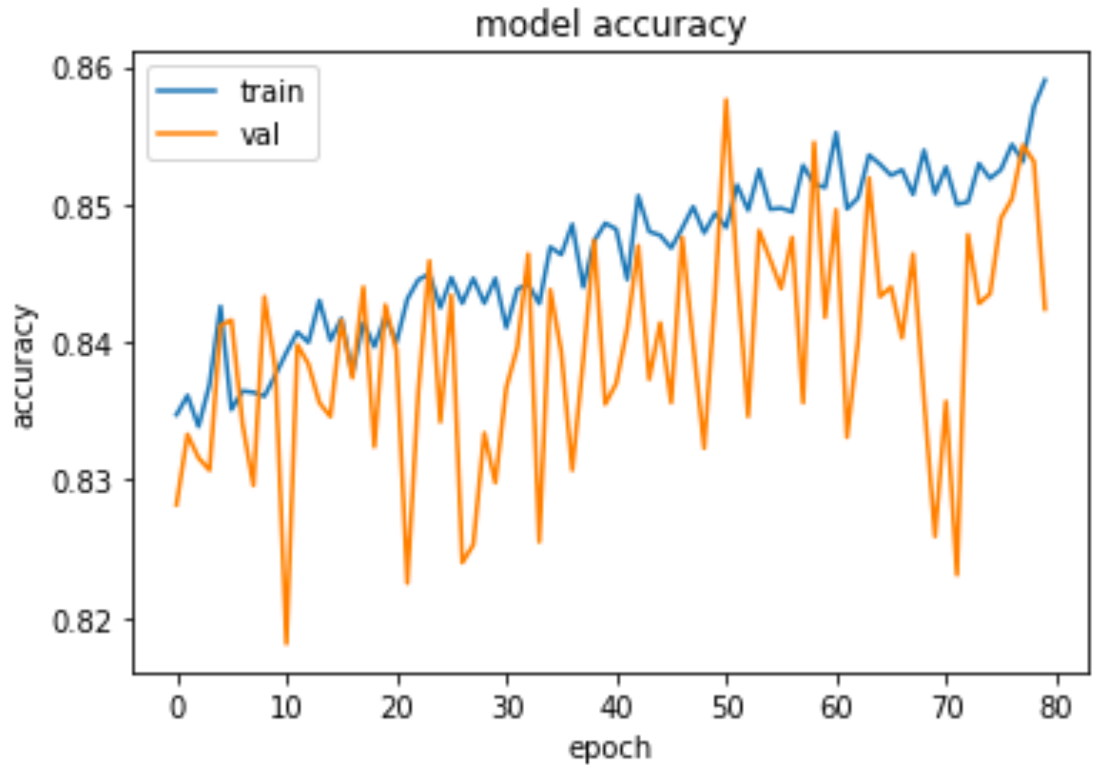
```
from keras.layers import *
from keras.models import *
from keras.callbacks import ModelCheckpoint
from keras.optimizers import SGD

checkpointer = ModelCheckpoint(filepath='cifa_10_zq.hdf5',
                               verbose=1, save_best_only=True) #save the best model

model_2.compile(optimizer = 'adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

epochs = 80

history_2 = model_2.fit_generator(train_generator,
          validation_data = (x_val,y_val),
          epochs=epochs,
          callbacks=[checkpointer],
          verbose=1)
```

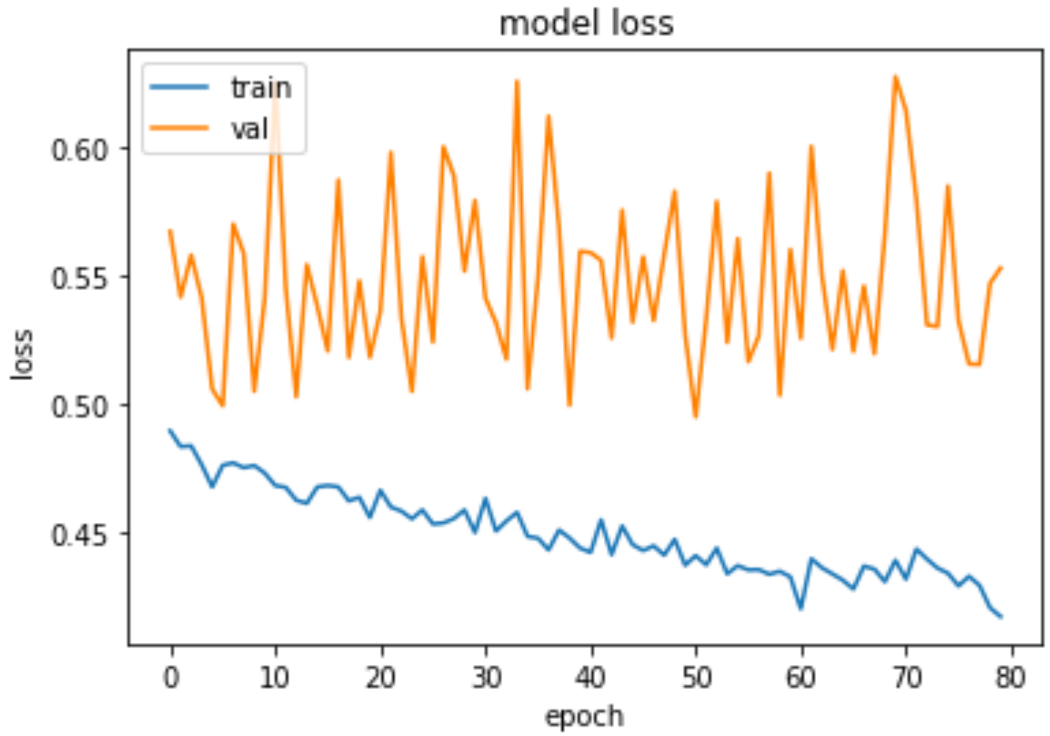
Datagen flow generates batches of 40,000 training images and this data is looped. Here 40,000/64(epochs) = 625 per epoch

Optimizer used – Adam
Metric – Accuracy
Loss – Categorical crossentropy
# of Epochs – 80
Batch size - 64

OUTPUT

```
Epoch 1/80
625/625 [==============================] - 24s 39ms/step - loss: 0.4894 - acc: 0.8347 - val_loss: 0.5674 - val_acc: 0.8282

Epoch 00001: val_loss improved from inf to 0.56741, saving model to cifa_10_zq.hdf5
Epoch 2/80
625/625 [==============================] - 21s 34ms/step - loss: 0.4832 - acc: 0.8361 - val_loss: 0.5417 - val_acc: 0.8333

Epoch 00002: val_loss improved from 0.56741 to 0.54170, saving model to cifa_10_zq.hdf5
Epoch 3/80
625/625 [==============================] - 21s 34ms/step - loss: 0.4835 - acc: 0.8339 - val_loss: 0.5581 - val_acc: 0.8316

Epoch 00003: val_loss did not improve from 0.54170
Epoch 4/80
625/625 [==============================] - 21s 34ms/step - loss: 0.4761 - acc: 0.8369 - val_loss: 0.5414 - val_acc: 0.8307

Epoch 00004: val_loss improved from 0.54170 to 0.54143, saving model to cifa_10_zq.hdf5
Epoch 5/80
625/625 [==============================] - 21s 34ms/step - loss: 0.4675 - acc: 0.8426 - val_loss: 0.5057 - val_acc: 0.8412

Epoch 00005: val_loss improved from 0.54143 to 0.50572, saving model to cifa_10_zq.hdf5
Epoch 6/80
625/625 [==============================] - 21s 34ms/step - loss: 0.4759 - acc: 0.8351 - val_loss: 0.4992 - val_acc: 0.8416

Epoch 00006: val_loss improved from 0.50572 to 0.49920, saving model to cifa_10_zq.hdf5
Epoch 7/80
625/625 [==============================] - 22s 35ms/step - loss: 0.4768 - acc: 0.8365 - val_loss: 0.5703 - val_acc: 0.8341

Epoch 00007: val_loss did not improve from 0.49920
Epoch 8/80
625/625 [==============================] - 22s 35ms/step - loss: 0.4749 - acc: 0.8364 - val_loss: 0.5587 - val_acc: 0.8296

Epoch 00008: val_loss did not improve from 0.49920
Epoch 9/80
625/625 [==============================] - 21s 34ms/step - loss: 0.4758 - acc: 0.8361 - val_loss: 0.5048 - val_acc: 0.8433

Epoch 00009: val_loss did not improve from 0.49920
Epoch 10/80
625/625 [==============================] - 22s 34ms/step - loss: 0.4728 - acc: 0.8376 - val_loss: 0.5413 - val_acc: 0.8380

Epoch 00010: val_loss did not improve from 0.49920
Epoch 11/80
625/625 [==============================] - 22s 35ms/step - loss: 0.4680 - acc: 0.8392 - val_loss: 0.6258 - val_acc: 0.8181
```


model accuracy
train
val
accuracy
0.86
0.85
0.84
0.83
0.82
0
10
20
30
40
50
60
70
80
epoch


model loss
train
val
loss
0.60
0.55
0.50
0.45
0
10
20
30
40
50
60
70
80
epoch

GRAPHS

```
[ ]  scores = model_2.evaluate(x_test, y_test, verbose=1)

     print('Test loss:', scores[0])
     print('Test accuracy:', scores[1])
```

```
10000/10000 [==============================] - 2s 157us/step
Test loss: 0.5057612937569618
Test accuracy: 0.8502
```

FINAL ACCURACY ON TEST DATA - 85.02%

SECOND CNN MODEL

```
#creating model
#Convolutional layer with Rectified linear unit(Relu) activation
#32 convolution filters used each of size 3x3 and getting output same as input shape by doing padding (32,3
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',kernel_initializer='he_uniform',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3),padding='same',kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

#Convolutional layer with Rectified linear unit(Relu) activation
#64 convolution filters used each of size 3x3 and getting output same as input shape by doing padding (32,3
model.add(Conv2D(64, (3, 3), padding='same',kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3),kernel_initializer='he_uniform',padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

#Convolutional layer with Rectified linear unit(Relu) activation
#128 convolution filters used each of size 3x3 and getting output same as input shape by doing padding (32,
model.add(Conv2D(128, (3, 3), padding='same',kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3),kernel_initializer='he_uniform',padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

#Flatten since too many dimensions, we only want a classification output.
model.add(Flatten())

#Fully Connected to get all relevant data.
model.add(Dense(512,kernel_initializer='he_uniform'))
model.add(Activation('relu'))


#One more dropout for convergence' sake.
model.add(Dropout(0.5))
model.add(Dense(num_classes))

#output a Softmax to squash the matrix into output probabilities.
model.add(Activation('softmax'))

plot_model(model,to_file="model.png",show_shapes=True,rankdir='TB',expand_nested=True)
```

6 convolutional layer

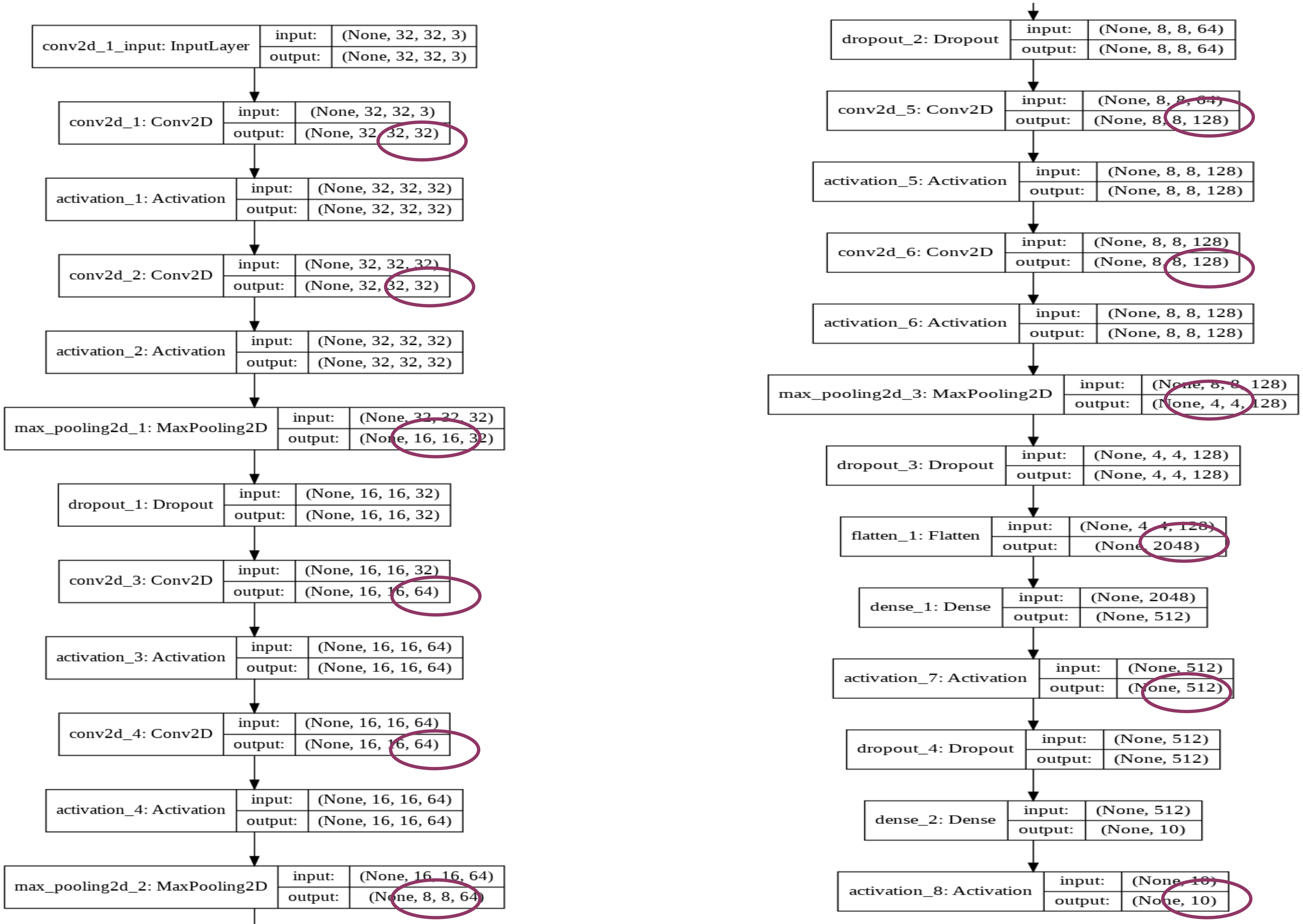
1. 2-32 filters with size (3,3)
   a. 1 Dropout 25% of nodes
2. 2-64 filters with size (3,3)
   a. 1 Dropout 25% of nodes
3. 2-128 filters with size (3,3)
   a. 1 Dropout 25% of nodes

Flattening

Reducing nodes to 512

Dropout of 50% nodes

Softmax layer with output 10 classifiers


conv2d_1_input: InputLayer | input: (None, 32, 32, 3) | output: (None, 32, 32, 3)
conv2d_1: Conv2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 32)
activation_1: Activation | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
conv2d_2: Conv2D | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
activation_2: Activation | input: (None, 32, 32, 32) | output: (None, 32, 32, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 32, 32, 32) | output: (None, 16, 16, 32)
dropout_1: Dropout | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)
conv2d_3: Conv2D | input: (None, 16, 16, 32) | output: (None, 16, 16, 64)
activation_3: Activation | input: (None, 16, 16, 64) | output: (None, 16, 16, 64)
conv2d_4: Conv2D | input: (None, 16, 16, 64) | output: (None, 16, 16, 64)
activation_4: Activation | input: (None, 16, 16, 64) | output: (None, 16, 16, 64)
max_pooling2d_2: MaxPooling2D | input: (None, 16, 16, 64) | output: (None, 8, 8, 64)
dropout_2: Dropout | input: (None, 8, 8, 64) | output: (None, 8, 8, 64)
conv2d_5: Conv2D | input: (None, 8, 8, 64) | output: (None, 8, 8, 128)
activation_5: Activation | input: (None, 8, 8, 128) | output: (None, 8, 8, 128)
conv2d_6: Conv2D | input: (None, 8, 8, 128) | output: (None, 8, 8, 128)
activation_6: Activation | input: (None, 8, 8, 128) | output: (None, 8, 8, 128)
max_pooling2d_3: MaxPooling2D | input: (None, 8, 8, 128) | output: (None, 4, 4, 128)
dropout_3: Dropout | input: (None, 4, 4, 128) | output: (None, 4, 4, 128)
flatten_1: Flatten | input: (None, 4, 4, 128) | output: (None, 2048)
dense_1: Dense | input: (None, 2048) | output: (None, 512)
activation_7: Activation | input: (None, 512) | output: (None, 512)
dropout_4: Dropout | input: (None, 512) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 10)
activation_8: Activation | input: (None, 10) | output: (None, 10)

SECOND CNN MODEL

```
[ ] # Let's train the model using RMSprop

    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name tf.train.Optimizer is deprecated. Please use tf.c

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3576: The name tf.log is deprecated. Please use
```

```
[ ] batch_size = 32
    num_classes = 10
    epochs = 70
    #data_augmentation = True
    num_predictions = 20

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

Optimizer used – SGD
Metric – Accuracy
Loss – Categorical crossentropy
# of Epochs – 70
Batch size - 32

OUTPUT

```
50000/50000 [==============================] - 406s 8ms/step - loss: 1.9535 - acc: 0.2695 - val_loss: 1.6470 - val_acc: 0.4006
Epoch 2/70
50000/50000 [==============================] - 404s 8ms/step - loss: 1.5781 - acc: 0.4170 - val_loss: 1.4046 - val_acc: 0.4833
Epoch 3/70
50000/50000 [==============================] - 405s 8ms/step - loss: 1.4160 - acc: 0.4788 - val_loss: 1.2569 - val_acc: 0.5482
Epoch 4/70
50000/50000 [==============================] - 405s 8ms/step - loss: 1.3075 - acc: 0.5243 - val_loss: 1.1615 - val_acc: 0.5845
Epoch 5/70
50000/50000 [==============================] - 405s 8ms/step - loss: 1.2118 - acc: 0.5597 - val_loss: 1.0902 - val_acc: 0.6136
Epoch 6/70
50000/50000 [==============================] - 406s 8ms/step - loss: 1.1380 - acc: 0.5909 - val_loss: 1.0006 - val_acc: 0.6532
Epoch 7/70
50000/50000 [==============================] - 406s 8ms/step - loss: 1.0769 - acc: 0.6120 - val_loss: 0.9672 - val_acc: 0.6570
Epoch 8/70
50000/50000 [==============================] - 406s 8ms/step - loss: 1.0297 - acc: 0.6300 - val_loss: 1.0318 - val_acc: 0.6334
Epoch 9/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.9936 - acc: 0.6448 - val_loss: 0.8717 - val_acc: 0.6859
Epoch 10/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.9522 - acc: 0.6606 - val_loss: 0.8669 - val_acc: 0.6941
Epoch 11/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.9189 - acc: 0.6723 - val_loss: 0.8621 - val_acc: 0.6946
Epoch 12/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.8905 - acc: 0.6827 - val_loss: 0.7994 - val_acc: 0.7215
Epoch 13/70
50000/50000 [==============================] - 407s 8ms/step - loss: 0.8666 - acc: 0.6912 - val_loss: 0.7706 - val_acc: 0.7306
Epoch 14/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.8412 - acc: 0.6998 - val_loss: 0.7662 - val_acc: 0.7320
Epoch 15/70
50000/50000 [==============================] - 407s 8ms/step - loss: 0.8218 - acc: 0.7074 - val_loss: 0.7638 - val_acc: 0.7297
Epoch 16/70
50000/50000 [==============================] - 407s 8ms/step - loss: 0.7972 - acc: 0.7165 - val_loss: 0.7267 - val_acc: 0.7443
Epoch 17/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.7714 - acc: 0.7245 - val_loss: 0.7082 - val_acc: 0.7499
Epoch 18/70
50000/50000 [==============================] - 405s 8ms/step - loss: 0.7560 - acc: 0.7328 - val_loss: 0.7347 - val_acc: 0.7421
Epoch 19/70
50000/50000 [==============================] - 408s 8ms/step - loss: 0.7391 - acc: 0.7398 - val_loss: 0.6779 - val_acc: 0.7612
Epoch 20/70
50000/50000 [==============================] - 407s 8ms/step - loss: 0.7229 - acc: 0.7441 - val_loss: 0.6838 - val_acc: 0.7618
Epoch 21/70
50000/50000 [==============================] - 409s 8ms/step - loss: 0.7045 - acc: 0.7514 - val_loss: 0.6711 - val_acc: 0.7659
```

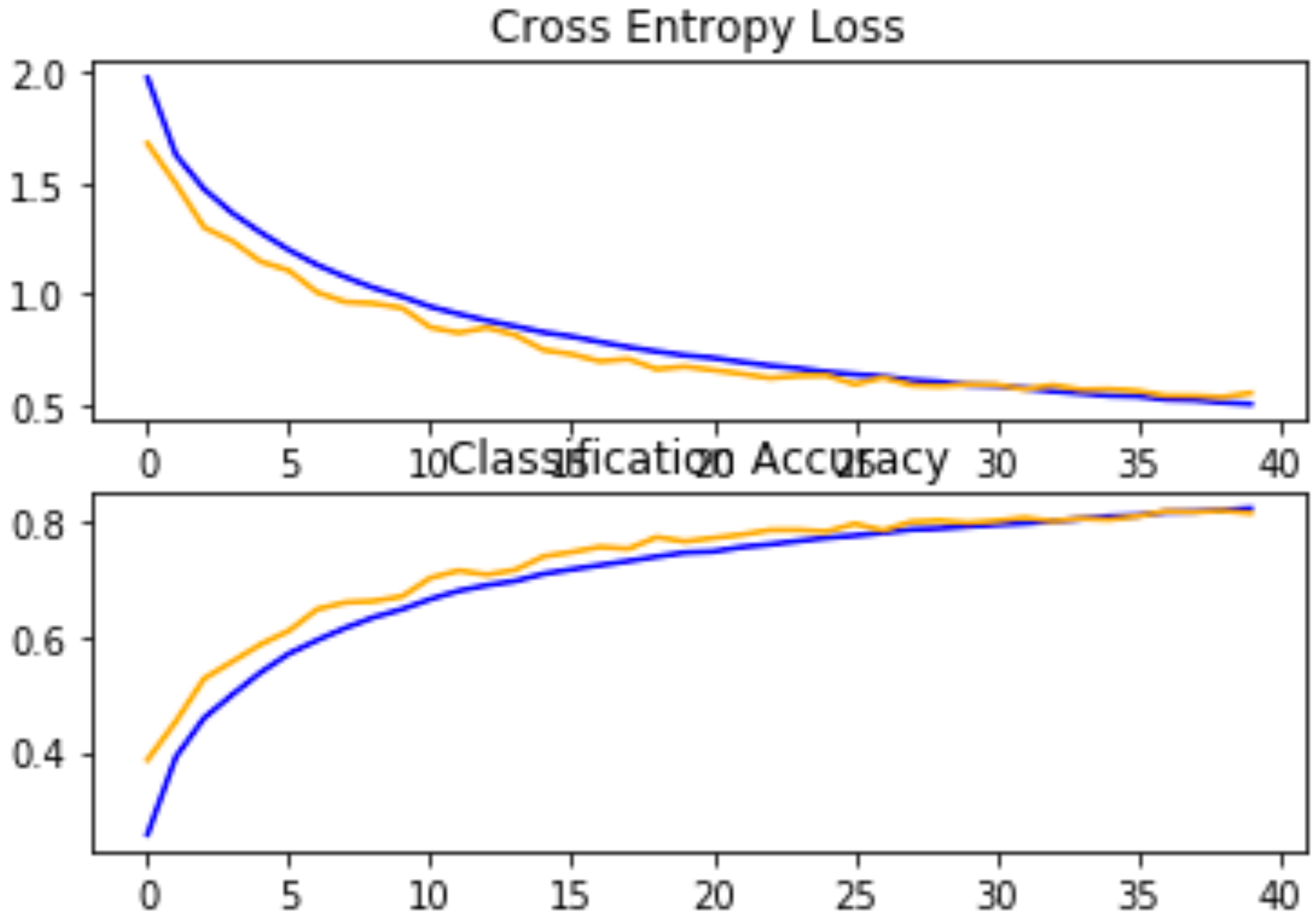
Continuously improving!

FINAL SCORE – 83.26%

```
Epoch 51/70
50000/50000 [==============================] - 404s 8ms/step - loss: 0.4239 - acc: 0.8471 - val_loss: 0.5146 - val_acc: 0.8275
Epoch 52/70
50000/50000 [==============================] - 405s 8ms/step - loss: 0.4159 - acc: 0.8518 - val_loss: 0.5169 - val_acc: 0.8272
Epoch 53/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.4082 - acc: 0.8538 - val_loss: 0.5268 - val_acc: 0.8264
Epoch 54/70
50000/50000 [==============================] - 408s 8ms/step - loss: 0.4044 - acc: 0.8549 - val_loss: 0.5149 - val_acc: 0.8274
Epoch 55/70
50000/50000 [==============================] - 407s 8ms/step - loss: 0.3984 - acc: 0.8580 - val_loss: 0.5198 - val_acc: 0.8251
Epoch 56/70
50000/50000 [==============================] - 408s 8ms/step - loss: 0.3949 - acc: 0.8574 - val_loss: 0.5329 - val_acc: 0.8216
Epoch 57/70
50000/50000 [==============================] - 409s 8ms/step - loss: 0.3867 - acc: 0.8629 - val_loss: 0.5057 - val_acc: 0.8332
Epoch 58/70
50000/50000 [==============================] - 409s 8ms/step - loss: 0.3812 - acc: 0.8637 - val_loss: 0.5051 - val_acc: 0.8284
Epoch 59/70
50000/50000 [==============================] - 408s 8ms/step - loss: 0.3755 - acc: 0.8660 - val_loss: 0.5060 - val_acc: 0.8323
Epoch 60/70
50000/50000 [==============================] - 410s 8ms/step - loss: 0.3719 - acc: 0.8675 - val_loss: 0.4926 - val_acc: 0.8365
Epoch 61/70
50000/50000 [==============================] - 410s 8ms/step - loss: 0.3673 - acc: 0.8673 - val_loss: 0.5330 - val_acc: 0.8288
Epoch 62/70
50000/50000 [==============================] - 409s 8ms/step - loss: 0.3695 - acc: 0.8684 - val_loss: 0.5075 - val_acc: 0.8322
Epoch 63/70
50000/50000 [==============================] - 410s 8ms/step - loss: 0.3542 - acc: 0.8732 - val_loss: 0.5174 - val_acc: 0.8316
Epoch 64/70
50000/50000 [==============================] - 411s 8ms/step - loss: 0.3546 - acc: 0.8744 - val_loss: 0.4993 - val_acc: 0.8374
Epoch 65/70
50000/50000 [==============================] - 409s 8ms/step - loss: 0.3472 - acc: 0.8746 - val_loss: 0.4990 - val_acc: 0.8375
Epoch 66/70
50000/50000 [==============================] - 405s 8ms/step - loss: 0.3427 - acc: 0.8779 - val_loss: 0.5052 - val_acc: 0.8355
Epoch 67/70
50000/50000 [==============================] - 405s 8ms/step - loss: 0.3385 - acc: 0.8787 - val_loss: 0.5355 - val_acc: 0.8334
Epoch 68/70
50000/50000 [==============================] - 404s 8ms/step - loss: 0.3348 - acc: 0.8805 - val_loss: 0.4991 - val_acc: 0.8391
Epoch 69/70
50000/50000 [==============================] - 405s 8ms/step - loss: 0.3343 - acc: 0.8800 - val_loss: 0.4958 - val_acc: 0.8389
Epoch 70/70
50000/50000 [==============================] - 406s 8ms/step - loss: 0.3241 - acc: 0.8834 - val_loss: 0.5150 - val_acc: 0.8326
```

Continuously improving!

OUTPUT GRAPHS



COMPARISON FOR DIFFERENT MODELS

# of CNN Layers	# of filters for CNN layer	Dropout for filters	# of Epochs	Batch Size	Flattening	Flattening dropout	Optimizer used	Accuracy
4	32,64	0.25,0.25	40	16	128,64	0.5	SGD	79.53%
4	32,64	0.25,0.25	100	16	512	0.5	SGD	81.39%
6	32,64,128	0.25,0.25,0.25	40	32	512	0.5	SGD	81.42%
6	32,64,128	0.25,0.25,0.25	70	32	512	0.5	SGD	83.26%
6	64,128,26	NA,NA,0.25	80	64	NA	0.4	Adam	85.02%

LEARNING FROM ASSIGNMENT

Image classification using Keras Library

Creating deep layers by adding convolutional layers, filters, max pooling and flattening layers

Creating model using different optimizers, # of epochs and batch size

Deep learning is a continuous process and happen gradually and keeps improving


LEARNING IMAGES
SO HOT RIGHT NOW
imgflip.com