In [ ]:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Flatten, ZeroPadding2D, BatchNormalization, AveragePooling2D
from keras import regularizers
from keras.optimizers import SGD
from keras import losses
from keras import utils
from keras import callbacks
from keras import initializers
import keras
import h5py
```

In [ ]:

```
import numpy as np
from PIL import Image
import copy

# constants
CLASS = 200
TRAIN_PER_CLASS = 400
VAL_PER_CLASS = 100
TOTAL_SAMPLES = CLASS * (TRAIN_PER_CLASS + VAL_PER_CLASS)
COLOR_CHANNELS = 3
IMAGE_WIDTH = 64
IMAGE_HEIGHT = 64
TEST_SAMPLES = 10000

# read all of the word net id
wnids = [id.strip('\n') for id in open('/datasets/tmp/cg181fdn/tiny-imagenet-200/wnids.t
xt').readlines()]

# data will store all of the data
data = {}

# train data
data['train'] = {}
data['train']['data'] = np.ndarray(shape=(TRAIN_PER_CLASS * CLASS, IMAGE_WIDTH, IMAGE_HE
IGHT, COLOR_CHANNELS), dtype=np.uint8)
data['train']['target'] = np.ndarray(shape=(TRAIN_PER_CLASS * CLASS,), dtype=np.uint8)

# validation data
data['val'] = {}
data['val']['data'] = np.ndarray(shape=(VAL_PER_CLASS * CLASS, IMAGE_WIDTH, IMAGE_HEIGHT
, COLOR_CHANNELS), dtype=np.uint8)
data['val']['target'] = np.ndarray(shape=(VAL_PER_CLASS * CLASS,), dtype=np.uint8)

# validation data
data['test'] = {}
data['test']['data'] = np.ndarray(shape=(TEST_SAMPLES, IMAGE_WIDTH, IMAGE_HEIGHT, COLOR_
CHANNELS), dtype=np.uint8)
data['test']['target'] = np.ndarray(shape=(TEST_SAMPLES,), dtype=np.uint8)

# iterate through work net ids
print("storing training and validation:")
for i in range(len(wnids)):
    wnid = wnids[i]
    print("%s: %d / %d" % (wnid, i + 1, len(wnids)))
    for j in range(TRAIN_PER_CLASS):
        temp = []
        path = "/datasets/tmp/cg181fdn/tiny-imagenet-200/train/{0}/images/{0}_{1}.JPEG".
format(wnid, j)
        data['train']['data'][i * TRAIN_PER_CLASS + j] = np.array(Image.open(path).conve
rt('RGB'))
        data['train']['target'][i * TRAIN_PER_CLASS + j] = wnids.index(wnid)
    for j in range(TRAIN_PER_CLASS, TRAIN_PER_CLASS + VAL_PER_CLASS):
        temp = []
        path = "/datasets/tmp/cg181fdn/tiny-imagenet-200/train/{0}/images/{0}_{1}.JPEG".
format(wnid, j)
        data['val']['data'][i * VAL_PER_CLASS + j-TRAIN_PER_CLASS] = np.array(Image.open
(path).convert('RGB'))
        data['val']['target'][i * VAL_PER_CLASS + j-TRAIN_PER_CLASS] = wnids.index(wnid)

# get the validation data
print("storing testing:")
for i, line in enumerate(map(lambda s: s.strip(), open('/datasets/tmp/cg181fdn/tiny-imag
enet-200/val/val_annotations.txt'))):
    print("%d/%d" % (i+1, 10000))
    name, wnid = line.split('\t')[0:2]
    temp = []
    path = "/datasets/tmp/cg181fdn/tiny-imagenet-200/val/images/{0}".format(name)
    data['test']['data'][i] = np.array(Image.open(path).convert('RGB'))
    data['test']['target'][i] = wnids.index(wnid)
```

In [ ]:

```
class TestCallback(callbacks.Callback):
    def __init__(self, test_data):
        self.test_data = test_data

    def on_epoch_end(self, epoch, logs={}):
        x, y = self.test_data
        loss, acc = self.model.evaluate(x, y, verbose=0)
        print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))

def saveModel(myModel, modelPath, weightPath):
    # serialize model to JSON then store to file
    model_json = myModel.to_json()
    with open(modelPath, "w") as json_file:
        json_file.write(model_json)

    # serialize weights to HDF5
    myModel.save_weights(weightPath)
    print("Saved model to disk")

def loadModel(modelPath, weightPath):
    # load json and create model
    json_file = open(modelPath, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = keras.models.model_from_json(loaded_model_json)

    # load weights into new model
    loaded_model.load_weights(weightPath)
    print("Loaded model from disk")

    return loaded_model

def storeResult(filename, title, myHist, testAcc):
    file = open(filename, 'w')
    file.write(title)
    file.write("--- Training: acc/loss ---\n")
    for acc, loss in zip(myHist.history["acc"], myHist.history["loss"]):
        file.write(str(acc)+"$"+str(loss)+"\n")
    file.write("--- Validation: acc/loss ---\n")
    for acc, loss in zip(myHist.history["val_acc"], myHist.history["val_loss"]):
        file.write(str(acc)+"$"+str(loss)+"\n")
    file.write(testAcc)
    file.close()
```

In [ ]:

```
def vgg_like():
    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64,64,3)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(256, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-6,l2=1e-6)))
    model.add(Dropout(0.5))
    model.add(Dense(200, activation='softmax',
                   kernel_regularizer=regularizers.l1_l2(l1=1e-5,l2=1e-5)))

    return model
```

In [ ]:

```
def vgg_16():
    model = Sequential()

    model.add(ZeroPadding2D(padding=(1,1), input_shape=(64,64,3)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', name="conv1_1"))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', name="conv1_2"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), name="block1_pool"))

    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', name="conv2_1"
))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', name="conv2_2"
))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), name="block2_pool"))

    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu', name="conv3_1"
))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu', name="conv3_2"
))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu', name="conv3_3"
))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), name="block3_pool"))

    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu', name="conv4_1"
))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu', name="conv4_2"
))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu', name="conv4_3"
))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), name="block4_pool"))

    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu',
                     kernel_regularizer=regularizers.l1_l2(l1=1e-7,l2=1e-7), name="conv5
_1"))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu',
                     kernel_regularizer=regularizers.l1_l2(l1=1e-6,l2=1e-6), name="conv5
_2"))
    model.add(ZeroPadding2D(padding=(1,1)))
    model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu',
                     kernel_regularizer=regularizers.l1_l2(l1=1e-5,l2=1e-5), name="conv5
_3"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,2), name="block5_pool"))

    model.add(Flatten())
    model.add(Dense(4096, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-4,l2=1e-4), name="fc6"))
    model.add(Dropout(0.75, name="drop6"))
    model.add(Dense(4096, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-4,l2=1e-4), name="fc7"))
    model.add(Dropout(0.75, name="drop7"))
    model.add(Dense(200, activation='softmax', name="fc8"))

    return model
```

In [ ]:

```
def alexnet():
    model = Sequential()

    model.add(Conv2D(filters=64, kernel_size=(11, 11), activation='relu', input_shape=(6
4,64,3)))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(1,1)))

    model.add(Conv2D(filters=128, kernel_size=(7, 7), activation='relu'))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(1,1)))

    model.add(Conv2D(filters=192, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(1,1)))

    model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu'))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(1,1)))

    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    model.add(Dense(512, activation='relu'))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    model.add(Dense(200, activation='softmax'))
    model.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))

    return model
```

In [ ]:

```
def lenet():
    model = Sequential()

    model.add(Conv2D(20, (5, 5), padding='same', activation='relu', input_shape=(64,64,3
)))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Conv2D(50, (5, 5), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(500, activation='relu'))
    model.add(Dropout(0.9))
    model.add(Dense(200, activation='softmax'))

    return model
```

In [ ]:

```
def myModel1(pool, myActivation):
    model = Sequential()

    model.add(Conv2D(64, (2, 2), activation=myActivation, padding='same', input_shape=(6
4,64,3)))
    model.add(pool)
    model.add(BatchNormalization())
    model.add(Conv2D(64, (2, 2), activation=myActivation, padding='same'))
    model.add(BatchNormalization())
    model.add(pool)
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (2, 2), activation=myActivation, padding='same'))
    model.add(BatchNormalization())
    model.add(pool)
    model.add(Conv2D(128, (2, 2), activation=myActivation, padding='same'))
    model.add(BatchNormalization())
    model.add(pool)
    model.add(Dropout(0.25))

    model.add(Conv2D(256, (2, 2), activation=myActivation, padding='same'))
    model.add(BatchNormalization())
    model.add(pool)
    model.add(Conv2D(256, (2, 2), activation=myActivation, padding='same'))
    model.add(BatchNormalization())
    model.add(pool)
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation=myActivation, kernel_regularizer=regularizers.l1_l2(
l1=1e-6,l2=1e-6)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(200, activation='softmax', kernel_regularizer=regularizers.l1_l2(l1=
1e-5,l2=1e-5)))

    return model
```

In [ ]:

```
def myModel2():
    model = Sequential()

    model.add(Conv2D(64, (2, 2), activation='relu', padding='same', input_shape=(64,64,3
)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (2, 2), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (2, 2), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l1_l2(l1=1e-
6,l2=1e-6)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(200, activation='softmax', kernel_regularizer=regularizers.l1_l2(l1=
1e-5,l2=1e-5)))

    return model
```

In [ ]:

```
def myModel3():
    model = Sequential()

    model.add(Conv2D(64, (2, 2), activation='relu', padding='same', input_shape=(64,64,3
)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512, activation='relu', kernel_regularizer=regularizers.l1_l2(l1=1e-
6,l2=1e-6)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(200, activation='softmax', kernel_regularizer=regularizers.l1_l2(l1=
1e-5,l2=1e-5)))

    return model
```

In [ ]:

```
# get data
train_x = np.array(data['train']['data'])
train_y = np.array(data['train']['target'])
train_y = utils.to_categorical(train_y, num_classes=200)

val_x = np.array(data['val']['data'])
val_y = np.array(data['val']['target'])
val_y = utils.to_categorical(val_y, num_classes=200)

test_x = np.array(data['test']['data'])
test_y = np.array(data['test']['target'])
test_y = utils.to_categorical(test_y, num_classes=200)

myModel = myModel1(AveragePooling2D(pool_size=(2, 2)), 'relu')

# compile the model
op = keras.optimizers.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0
)
myModel.compile(loss='categorical_crossentropy', optimizer=op, metrics=['accuracy'])

# fit and evaluate
hist = myModel.fit(x=train_x, y=train_y, batch_size=128, validation_data=(val_x, val_y),
 shuffle=True,
                   initial_epoch=0, epochs=100,
                   callbacks=[TestCallback((test_x, test_y))])
testScore = myModel.evaluate(test_x, test_y, verbose=0)
testAccuracy = "Test Accuracy: %.2f%%" % (testScore[1]*100)
print(testAccuracy)
```

In [ ]:

```
resultFilename = "./Results/model1.txt"
storeResult(resultFilename, "model1 Adam {lr: 1e-4, epoch: 100}\n", hist, testAccuracy)
```

In [ ]:

```
# model and weight paths
modelPath = "./Models/model1-model Adam {epoch: 100}.json"
weightPath = "./Models/model1-model Adam {epoch: 100}.h5"

# store model
saveModel(myModel, modelPath, weightPath)
```

In [ ]:

```
# attempting transfer learning
op = keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
modelPath = "./Models/model1_avg-weight Adam {epoch: 100}.json"
weightPath = "./Models/model1_avg-model Adam {epoch: 100}.h5"

# load model
loadedModel = loadModel(modelPath, weightPath)

# evaluate loaded model on test data
loadedModel.compile(loss='categorical_crossentropy', optimizer=op, metrics=['accuracy'])
score = loadedModel.evaluate(test_x, test_y, verbose=0)
print("%s: %.2f%%" % (loadedModel.metrics_names[1], score[1]*100))
```

In [ ]:

```
# continue fitting
hist = myModel.fit(x=train_x, y=train_y, batch_size=32, validation_data=(val_x, val_y),
shuffle=True,
                   initial_epoch=100, epochs=200,
                   callbacks=[TestCallback((test_x, test_y))])
```