In [ ]:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

# Imports
import numpy as np
import tensorflow as tf
import gzip
import cPickle
```

In [ ]:

```
def load_zipped_pickle(filename):
    with gzip.open(filename, 'rb') as f:
        loaded_object = cPickle.load(f)
        return loaded_object
```

In [ ]:
```
def vgg_16(features, labels, mode):

    # Input Layer
    input_layer = tf.reshape(features["x"], [-1, 64, 64, 3])

    # Convolutional Layer 1
    conv1 = tf.layers.conv2d(inputs=input_layer, filters=64, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer 2
    conv2 = tf.layers.conv2d(inputs=conv1, filters=64, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Pooling Layer 1
    pool1 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

    # Convolutional Layer 3
    conv3 = tf.layers.conv2d(inputs=pool1, filters=128, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer 4
    conv4 = tf.layers.conv2d(inputs=conv3, filters=128, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Pooling Layer 2
    pool2 = tf.layers.max_pooling2d(inputs=conv4, pool_size=[2, 2], strides=2)

    # Convolutional Layer 5
    conv5 = tf.layers.conv2d(inputs=pool2, filters=256, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer 6
    conv6 = tf.layers.conv2d(inputs=conv5, filters=256, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer 7
    conv7 = tf.layers.conv2d(inputs=conv6, filters=256, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Pooling Layer 3
    pool3 = tf.layers.max_pooling2d(inputs=conv7, pool_size=[2, 2], strides=2)

    # Convolutional Layer 8
    conv8 = tf.layers.conv2d(inputs=pool3, filters=512, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer 9
    conv9 = tf.layers.conv2d(inputs=conv8, filters=512, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer 10
    conv10 = tf.layers.conv2d(inputs=conv9, filters=512, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Pooling Layer 4
    pool4 = tf.layers.max_pooling2d(inputs=conv10, pool_size=[2, 2], strides=2)

    # Convolutional Layer 11
    conv11 = tf.layers.conv2d(inputs=pool4, filters=512, kernel_size=[3, 3], padding="same", activation=tf.nn.relu,\
                              kernel_regularizer=tf.keras.regularizers.l1_l2(0.0000001,0.0000001),\
                              activity_regularizer=tf.keras.regularizers.l1_l2(0.0000001,0.0000001))

    # Convolutional Layer 12
    conv12 = tf.layers.conv2d(inputs=conv11, filters=512, kernel_size=[3, 3], padding="same", activation=tf.nn.relu,\
                              kernel_regularizer=tf.keras.regularizers.l1_l2(0.000001,0.000001),\
                              activity_regularizer=tf.keras.regularizers.l1_l2(0.000001,0.000001))

    # Convolutional Layer 13
    conv13 = tf.layers.conv2d(inputs=conv12, filters=512, kernel_size=[3, 3], padding="same", activation=tf.nn.relu,\
                              kernel_regularizer=tf.keras.regularizers.l1_l2(0.00001,0.00001),\
                              activity_regularizer=tf.keras.regularizers.l1_l2(0.00001,0.00001))

    # Pooling Layer 5
    pool5 = tf.layers.max_pooling2d(inputs=conv13, pool_size=[2, 2], strides=2)
    pool5_flat = tf.contrib.layers.flatten(pool5)

    # dense 1
    dense1 = tf.layers.dense(inputs=pool5_flat, units=4096, activation=tf.nn.relu,\
                             kernel_initializer=tf.contrib.layers.xavier_initializer(),\
                             kernel_regularizer=tf.keras.regularizers.l1_l2(0.0001,0.0001),\
                             activity_regularizer=tf.keras.regularizers.l1_l2(0.00001,0.00001))
    dropout1 = tf.layers.dropout(inputs=dense1, rate=0.75)

    # dense 2
    dense2 = tf.layers.dense(inputs=dropout1, units=4096, activation=tf.nn.relu,\
                             kernel_initializer=tf.contrib.layers.xavier_initializer(),\
                             kernel_regularizer=tf.keras.regularizers.l1_l2(0.0001,0.0001),\
                             activity_regularizer=tf.keras.regularizers.l1_l2(0.00001,0.00001))
    dropout2 = tf.layers.dropout(inputs=dense2, rate=0.75)

    # dense 3
    logits = tf.layers.dense(inputs=dropout2, units=200, activation=tf.nn.relu,\
                            kernel_initializer=tf.contrib.layers.xavier_initializer())

    predictions = {
        # Generate predictions (for PREDICT and EVAL mode)
        "classes": tf.argmax(input=logits, axis=1),
        # Add `softmax_tensor` to the graph. It is used for PREDICT and by the
        # `logging_hook`.
        "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
    }

    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

    # Calculate Loss (for both TRAIN and EVAL modes)
    onehot_labels = tf.one_hot(indices=tf.cast(labels, tf.int32), depth=200)
    loss = tf.losses.softmax_cross_entropy(onehot_labels=onehot_labels, logits=logits)

    # Configure the Training Op (for TRAIN mode)
    if mode == tf.estimator.ModeKeys.TRAIN:
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
        train_op = optimizer.minimize(loss=loss, global_step=tf.train.get_global_step())
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

    # Add evaluation metrics (for EVAL mode)
    eval_metric_ops = {"accuracy": tf.metrics.accuracy(labels=labels, predictions=predictions["classes"])}
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)
```

In [ ]:

```
def cnn_model2_fn(features, labels, mode):
    """Model function for CNN."""
    # Input Layer
    input_layer = tf.reshape(features["x"], [-1, 64, 64, 3])

    # Convolutional Layer #1
    conv1 = tf.layers.conv2d(inputs=input_layer, filters=32, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer #2
    conv2 = tf.layers.conv2d(inputs=conv1, filters=32, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Pooling Layer #1
    pool1 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

    # dropout #1
    #dropout1 = tf.layers.dropout(inputs=pool1, rate=0.25, training=mode == tf.estimator.ModeKeys.TRAIN)
    dropout1 = tf.layers.dropout(inputs=pool1, rate=0.25)

    # Convolutional Layer #3
    conv3 = tf.layers.conv2d(inputs=dropout1, filters=64, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Convolutional Layer #4
    conv4 = tf.layers.conv2d(inputs=conv3, filters=64, kernel_size=[3, 3], padding="same", activation=tf.nn.relu)

    # Pooling Layer #2
    pool2 = tf.layers.max_pooling2d(inputs=conv4, pool_size=[2, 2], strides=2)
    pool2_flat = tf.contrib.layers.flatten(pool2)

    # dropout #2
    dropout2 = tf.layers.dropout(inputs=pool2_flat, rate=0.25)

    # dense #1
    dense1 = tf.layers.dense(inputs=dropout2, units=256, activation=tf.nn.relu)
    # dropout #3
    dropout3 = tf.layers.dropout(inputs=dense1, rate=0.5)

    # dense #2
    logits = tf.layers.dense(inputs=dropout3, units=200, activation=tf.nn.relu)

    predictions = {
        # Generate predictions (for PREDICT and EVAL mode)
        "classes": tf.argmax(input=logits, axis=1),
        # Add `softmax_tensor` to the graph. It is used for PREDICT and by the
        # `logging_hook`.
        "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
    }

    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)
    # Calculate Loss (for both TRAIN and EVAL modes)
    onehot_labels = tf.one_hot(indices=tf.cast(labels, tf.int32), depth=200)
    loss = tf.losses.softmax_cross_entropy(onehot_labels=onehot_labels, logits=logits)

    # Configure the Training Op (for TRAIN mode)
    if mode == tf.estimator.ModeKeys.TRAIN:
        #optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
        optimizer = tf.keras.optimizers.SGD(lr=0.001, decay=0.000001, momentum=0.9, nesterov=True)
        train_op = optimizer.minimize(loss=loss, global_step=tf.train.get_global_step())
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train_op=train_op)

    # Add evaluation metrics (for EVAL mode)
    eval_metric_ops = {"accuracy": tf.metrics.accuracy(labels=labels, predictions=predictions["classes"])}
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)
```

In [ ]:

```
tf.logging.set_verbosity(tf.logging.INFO)

# load data
print("loading data from file...")
data = load_zipped_pickle("tinyImageData")
train_data = np.array(data["train"]["data"], dtype = np.float16)
train_labels = np.array(data["train"]["target"], dtype = np.float16)
val_data = np.array(data["val"]["data"], dtype = np.float16)
val_labels = np.array(data["val"]["target"], dtype = np.float16)
```

In [ ]:

```
# Create the Estimator
tiny_imagenet_clf = tf.estimator.Estimator(model_fn=vgg_16, model_dir="tiny_imagenet_cnn_model")

# Set up logging for predictions
# Log the values in the "Softmax" tensor with label "probabilities"
tensors_to_log = {"probabilities": "softmax_tensor"}
logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log, every_n_iter=500)

# Train the model
train_input_fn = tf.estimator.inputs.numpy_input_fn(x={"x": train_data}, y=train_labels, batch_size=256,
                                                    num_epochs=6, shuffle=True)
tiny_imagenet_clf.train(input_fn=train_input_fn, steps=20000, hooks=[logging_hook])

# Evaluate the model and print results
eval_input_fn = tf.estimator.inputs.numpy_input_fn(x={"x": val_data}, y=val_labels, num_epochs=1, shuffle=False)
eval_results = tiny_imagenet_clf.evaluate(input_fn=eval_input_fn)
print(eval_results)
```

In [ ]: