To read networks from TensorFlow framework there is cv::dnn::readNetFromTensorflow method which can work with `.pb` files with frozen TensorFlow graph. However sometimes it is not enough to have only `.pb` file to import network into OpenCV.

Depends on topology, graph may contains some unfused layers which are not covered by internal subgraphs fusion procedure. In example, instead of a single `Flatten` node for tensors flattening (reshaping from 4D `[N, C, H, W]` to 2D `[N, C*H*W]`) graph will have a set of nodes that precompute final shape and then call `Reshape` operation:

If you know how to replace TensorFlow subgraph to something simpler which can be parser by OpenCV (see tf_importer.cpp and tf_graph_simplifier.cpp) you may use **optional** parameter of cv::dnn::readNetFromTensorflow to use a text graph representation of a frozen `.pb` file.

Whet is text graph representation? TensorFlow uses Google Protobuf format to store deep learning networks. This format can be as binary as textual. Due models usually have a lot of trainable weights it is more optimal to store them as binary rather than text file.

To generate `.pbtxt` file for OpenCV, you need to convert a `.pb` model to text file. The following script can do it:

```
import tensorflow as tf

pb_file = 'model.pb'
graph_def = tf.compat.v1.GraphDef()

try:
    with tf.io.gfile.GFile(pb_file, 'rb') as f:
        graph_def.ParseFromString(f.read())
except:
    with tf.gfile.FastGFile(pb_file, 'rb') as f:
        graph_def.ParseFromString(f.read())

# Delete weights
for i in reversed(range(len(graph_def.node))):
    if graph_def.node[i].op == 'Const':
        del graph_def.node[i]

graph_def.library.Clear()

tf.compat.v1.train.write_graph(graph_def, "", 'model.pbtxt', as_text=True)
```

Then open `model.pbtxt` and replace a problematic subgraph

```
node {
  name: "flatten_1/Shape"
  op: "Shape"
```

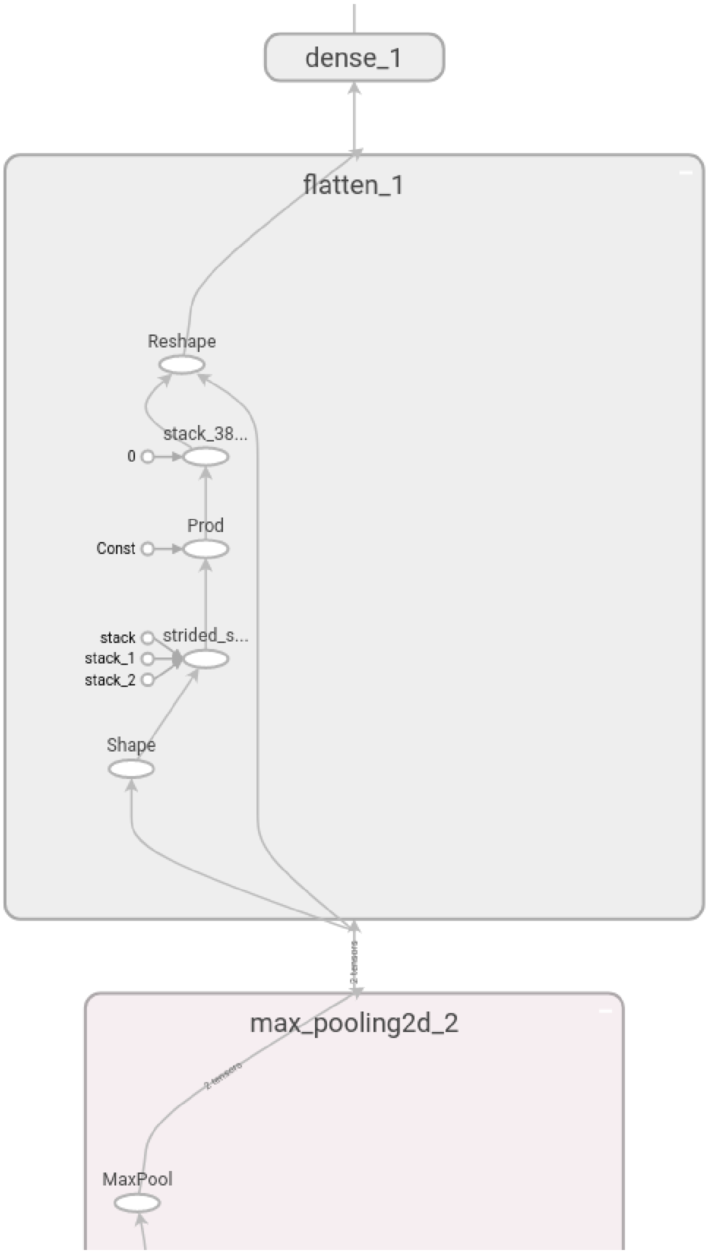

Figure 1: image

```
  input: "max_pooling2d_2/MaxPool"
  attr {
    key: "T"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "out_type"
    value {
      type: DT_INT32
    }
  }
}
node {
  name: "flatten_1/strided_slice"
  op: "StridedSlice"
  input: "flatten_1/Shape"
  input: "flatten_1/strided_slice/stack"
  input: "flatten_1/strided_slice/stack_1"
  input: "flatten_1/strided_slice/stack_2"
  attr {
    key: "Index"
    value {
      type: DT_INT32
    }
  }
  attr {
    key: "T"
    value {
      type: DT_INT32
    }
  }
  attr {
    key: "begin_mask"
    value {
      i: 0
    }
  }
  attr {
    key: "ellipsis_mask"
    value {
      i: 0
    }
  }
  attr {
    key: "end_mask"
    value {
      i: 1
    }
  }
  attr {
    key: "new_axis_mask"
    value {
      i: 0
    }
  }
  attr {
    key: "shrink_axis_mask"
    value {
      i: 0
    }
  }
}
node {
  name: "flatten_1/Prod"
  op: "Prod"
  input: "flatten_1/strided_slice"
  input: "flatten_1/Const"
  attr {
    key: "T"
    value {
      type: DT_INT32
    }
  }
  attr {
    key: "Tidx"
    value {
      type: DT_INT32
    }
  }
  attr {
    key: "keep_dims"
    value {
      b: false
    }
  }
}
node {
  name: "flatten_1/stack_3862"
  op: "Pack"
  input: "flatten_1/stack_3862/0"
  input: "flatten_1/Prod"
  attr {
    key: "N"
    value {
      i: 2
    }
  }
  attr {
    key: "T"
    value {
      type: DT_INT32
    }
  }
  attr {
    key: "axis"
    value {
      i: 0
    }
  }
}
node {
  name: "flatten_1/Reshape"
  op: "Reshape"
  input: "max_pooling2d_2/MaxPool"
  input: "flatten_1/stack_3862"
  attr {
    key: "T"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "Tshape"
    value {
      type: DT_INT32
    }
  }
}
```

to a single `Flatten` layer which can be easily imported by OpenCV:

```
node {
  name: "flatten_1/Reshape"
  op: "Flatten"
  input: "max_pooling2d_2/MaxPool"
}
```

NOTE: check that after removing the nodes connections between nodes are not broken. That's why we use the same name for new `Flatten` node because there is a consumer which is connected to this specific name:

```
node {
  name: "dense_1/MatMul"
  op: "MatMul"
  input: "flatten_1/Reshape"
  input: "dense_1/MatMul/ReadVariableOp"
  attr {
    key: "T"
    value { type: DT_FLOAT }
  }
  attr {
    key: "transpose_a"
    value { b: false }
  }
  attr {
    key: "transpose_b"
    value { b: false }
  }
}
```

Finally, use both `.pb` and `.pbtxt` to import network into OpenCV:

```
import cv2 as cv

net = cv.dnn.readNetFromTensorflow('model.pb', 'model.pbtxt')
```