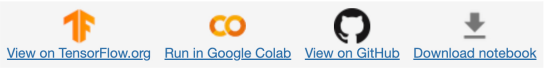
In [92]:
```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```



Sparsity and cluster preserving quantization aware training (PCQAT) Keras example

Overview

This is an end to end example showing the usage of the **sparsity and cluster preserving quantization aware training (PCQAT)** API, part of the TensorFlow Model Optimization Toolkit's collaborative optimization pipeline.

Other pages

For an introduction to the pipeline and other available techniques, see the collaborative optimization overview page.

Contents

In the tutorial, you will:

1. Train a `keras` model for the MNIST dataset from scratch.
2. Fine-tune the model with pruning and see the accuracy and observe that the model was successfully pruned.
3. Apply sparsity preserving clustering on the pruned model and observe that the sparsity applied earlier has been preserved.
4. Apply QAT and observe the loss of sparsity and clusters.
5. Apply PCQAT and observe that both sparsity and clustering applied earlier have been preserved.
6. Generate a TFLite model and observe the effects of applying PCQAT on it.
7. Compare the sizes of the different models to observe the compression benefits of applying sparsity followed by the collaborative optimization techniques of sparsity preserving clustering and PCQAT.
8. Compare the accurracy of the fully optimized model with the un-optimized baseline model accuracy.

Setup

You can run this Jupyter Notebook in your local virtualenv or colab. For details of setting up dependencies, please refer to the installation guide.

In [93]:
```
! pip install -q tensorflow-model-optimization
```

In [94]:
```
import tensorflow as tf
import tf_keras as keras

import numpy as np
import tempfile
import zipfile
import os
```

Train a keras model for MNIST to be pruned and clustered

In [95]:
```
# Load MNIST dataset
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the input image so that each pixel value is between 0 to 1.
train_images = train_images / 255.0
test_images  = test_images / 255.0

model = keras.Sequential([
  keras.layers.InputLayer(input_shape=(28, 28)),
  keras.layers.Reshape(target_shape=(28, 28, 1)),
  keras.layers.Conv2D(filters=12, kernel_size=(3, 3),
                      activation=tf.nn.relu),
  keras.layers.MaxPooling2D(pool_size=(2, 2)),
  keras.layers.Flatten(),
  keras.layers.Dense(10)
])

opt = keras.optimizers.Adam(learning_rate=1e-3)

# Train the digit classification model
```

```
model.compile(optimizer=opt,
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(
    train_images,
    train_labels,
    validation_split=0.1,
    epochs=10
)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use th
e legacy TF-Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
```

```
Epoch 1/10
1688/1688 [==============================] - 5s 3ms/step - loss: 0.3009 - accuracy: 0.9154 - val_loss: 0.1184 - val
_accuracy: 0.9693
Epoch 2/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.1149 - accuracy: 0.9670 - val_loss: 0.0804 - val
_accuracy: 0.9798
Epoch 3/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0849 - accuracy: 0.9755 - val_loss: 0.0775 - val
_accuracy: 0.9798
Epoch 4/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0698 - accuracy: 0.9796 - val_loss: 0.0673 - val
_accuracy: 0.9828
Epoch 5/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0619 - accuracy: 0.9814 - val_loss: 0.0604 - val
_accuracy: 0.9855
Epoch 6/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0542 - accuracy: 0.9839 - val_loss: 0.0579 - val
_accuracy: 0.9860
Epoch 7/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0493 - accuracy: 0.9853 - val_loss: 0.0599 - val
_accuracy: 0.9840
Epoch 8/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0451 - accuracy: 0.9866 - val_loss: 0.0584 - val
_accuracy: 0.9863
Epoch 9/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0407 - accuracy: 0.9876 - val_loss: 0.0585 - val
_accuracy: 0.9848
Epoch 10/10
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0369 - accuracy: 0.9886 - val_loss: 0.0593 - val
_accuracy: 0.9848
```

Out[95]:
```
<tf_keras.src.callbacks.History at 0x2c5f2b950>
```

In [96]:
```
def get_gzipped_model_size(model):
  with tempfile.NamedTemporaryFile(suffix=".h5") as temp_file:
    model.save(temp_file.name)

    _, zipped_file = tempfile.mkstemp('.zip')
    with zipfile.ZipFile(zipped_file, 'w', compression=zipfile.ZIP_DEFLATED) as f:
      f.write(temp_file.name)

    x = os.path.getsize(zipped_file)

    os.remove(zipped_file)

    return x / 1000
```

Evaluate the baseline model and save it for later usage

In [97]:
```
_, baseline_model_accuracy = model.evaluate(
    test_images, test_labels, verbose=0)

print('Baseline test accuracy:', baseline_model_accuracy)

sz = get_gzipped_model_size(model)
print("Base model size: ",  sz , ' KB' )
```

```
Baseline test accuracy: 0.9830999970436096
Base model size:  234.997  KB
```

```
/Users/ajaymaheshwari/anaconda3/lib/python3.11/site-packages/tf_keras/src/engine/training.py:3098: UserWarning: You
are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using
instead the native TF-Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Prune and fine-tune the model to 50% sparsity

Apply the `prune_low_magnitude()` API to achieve the pruned model that is to be clustered in the next step. Refer to the pruning comprehensive guide for more information on the pruning API.

Define the model and apply the sparsity API

Note that the pre-trained model is used.

In [98]:
```
import tensorflow_model_optimization as tfmot

prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

pruning_params = {
      'pruning_schedule': tfmot.sparsity.keras.ConstantSparsity(0.5, begin_step=0, frequency=100)
  }

callbacks = [
  tfmot.sparsity.keras.UpdatePruningStep()
```

```
  tfmot.sparsity.keras.UpdatePruningStep()
]

pruned_model = prune_low_magnitude(model, **pruning_params)

# Use smaller learning rate for fine-tuning
opt = keras.optimizers.Adam(learning_rate=1e-5)

pruned_model.compile(
  loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  optimizer=opt,
  metrics=['accuracy'])
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use th
e legacy TF-Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
```

Fine-tune the model, check sparsity, and evaluate the accuracy against baseline

Fine-tune the model with pruning for 3 epochs.

In [99]:
```
# Fine-tune model
pruned_model.fit(
  train_images,
  train_labels,
  epochs=3,
  validation_split=0.1,
  callbacks=callbacks)
```

```
Epoch 1/3
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0891 - accuracy: 0.9705 - val_loss: 0.1026 - val
_accuracy: 0.9673
Epoch 2/3
1688/1688 [==============================] - 4s 3ms/step - loss: 0.0732 - accuracy: 0.9767 - val_loss: 0.0906 - val
_accuracy: 0.9715
Epoch 3/3
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0649 - accuracy: 0.9792 - val_loss: 0.0846 - val
_accuracy: 0.9747
```

Out[99]:
```
<tf_keras.src.callbacks.History at 0x28f4ceb90>
```

Define helper functions to calculate and print the sparsity and clusters of the model.

In [100]:
```
def print_model_weights_sparsity(model):
    for layer in model.layers:
        if isinstance(layer, keras.layers.Wrapper):
            weights = layer.trainable_weights
        else:
            weights = layer.weights
        for weight in weights:
            if "kernel" not in weight.name or "centroid" in weight.name:
                continue
            weight_size = weight.numpy().size
            zero_num = np.count_nonzero(weight == 0)
            print(
                f"{weight.name}: {zero_num/weight_size:.2%} sparsity ",
                f"({zero_num}/{weight_size})",
            )

def print_model_weight_clusters(model):
    for layer in model.layers:
        if isinstance(layer, keras.layers.Wrapper):
            weights = layer.trainable_weights
        else:
            weights = layer.weights
        for weight in weights:
            # ignore auxiliary quantization weights
            if "quantize_layer" in weight.name:
                continue
            if "kernel" in weight.name:
                unique_count = len(np.unique(weight))
                print(
                    f"{layer.name}/{weight.name}: {unique_count} clusters "
                )
```

Let's strip the pruning wrapper first, then check that the model kernels were correctly pruned.

In [101]:
```
stripped_pruned_model = tfmot.sparsity.keras.strip_pruning(pruned_model)

print_model_weights_sparsity(stripped_pruned_model)
```

```
conv2d_6/kernel:0: 50.00% sparsity  (54/108)
dense_6/kernel:0: 50.00% sparsity  (10140/20280)
```

Apply sparsity preserving clustering and check its effect on model sparsity in both cases

Next, apply sparsity preserving clustering on the pruned model and observe the number of clusters and check that the sparsity is preserved.

In [102]:
```
import tensorflow_model_optimization as tfmot
from tensorflow_model_optimization.python.core.clustering.keras.experimental import (
    cluster,
)

cluster_weights = tfmot.clustering.keras.cluster_weights
CentroidInitialization = tfmot.clustering.keras.CentroidInitialization

cluster_weights = cluster.cluster_weights
```

```
clustering_params = {
  'number_of_clusters': 8,
  'cluster_centroids_init': CentroidInitialization.KMEANS_PLUS_PLUS,
  'preserve_sparsity': True
}

sparsity_clustered_model = cluster_weights(stripped_pruned_model, **clustering_params)

sparsity_clustered_model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

print('Train sparsity preserving clustering model:')
sparsity_clustered_model.fit(train_images, train_labels,epochs=3, validation_split=0.1)
```

```
Train sparsity preserving clustering model:
Epoch 1/3
1688/1688 [==============================] - 6s 3ms/step - loss: 0.0500 - accuracy: 0.9846 - val_loss: 0.0639 - val
_accuracy: 0.9818
Epoch 2/3
1688/1688 [==============================] - 6s 3ms/step - loss: 0.0451 - accuracy: 0.9857 - val_loss: 0.0616 - val
_accuracy: 0.9850
Epoch 3/3
1688/1688 [==============================] - 6s 3ms/step - loss: 0.0430 - accuracy: 0.9862 - val_loss: 0.0671 - val
_accuracy: 0.9837
```

Out[102]: `<tf_keras.src.callbacks.History at 0x28c36c9d0>`

Strip the clustering wrapper first, then check that the model is correctly pruned and clustered.

In [103]:
```
stripped_clustered_model = tfmot.clustering.keras.strip_clustering(sparsity_clustered_model)

print("Model sparsity:\n")
print_model_weights_sparsity(stripped_clustered_model)

print("\nModel clusters:\n")
print_model_weight_clusters(stripped_clustered_model)
```

```
Model sparsity:

kernel:0: 53.70% sparsity  (58/108)
kernel:0: 56.18% sparsity  (11394/20280)

Model clusters:

conv2d_6/kernel:0: 8 clusters
dense_6/kernel:0: 8 clusters
```

Apply QAT and PCQAT and check effect on model clusters and sparsity

Next, apply both QAT and PCQAT on the sparse clustered model and observe that PCQAT preserves weight sparsity and clusters in your model. Note that the stripped model is passed to the QAT and PCQAT API.

In [104]:
```
# QAT
qat_model = tfmot.quantization.keras.quantize_model(stripped_clustered_model)

qat_model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
print('Train qat model:')
qat_model.fit(train_images, train_labels, batch_size=128, epochs=1, validation_split=0.1)

# PCQAT
quant_aware_annotate_model = tfmot.quantization.keras.quantize_annotate_model(
              stripped_clustered_model)
pcqat_model = tfmot.quantization.keras.quantize_apply(
              quant_aware_annotate_model,
              tfmot.experimental.combine.Default8BitClusterPreserveQuantizeScheme(preserve_sparsity=True))

pcqat_model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
print('Train pcqat model:')
pcqat_model.fit(train_images, train_labels, batch_size=128, epochs=1, validation_split=0.1)
```

```
Train qat model:
422/422 [==============================] - 3s 7ms/step - loss: 0.0348 - accuracy: 0.9895 - val_loss: 0.0621 - val_a
ccuracy: 0.9862
Train pcqat model:
WARNING:tensorflow:Gradients do not exist for variables ['conv2d_6/kernel:0', 'dense_6/kernel:0'] when minimizing t
he loss. If you're using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['conv2d_6/kernel:0', 'dense_6/kernel:0'] when minimizing t
he loss. If you're using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['conv2d_6/kernel:0', 'dense_6/kernel:0'] when minimizing t
he loss. If you're using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['conv2d_6/kernel:0', 'dense_6/kernel:0'] when minimizing t
he loss. If you're using `model.compile()`, did you forget to provide a `loss` argument?
422/422 [==============================] - 4s 7ms/step - loss: 0.0350 - accuracy: 0.9890 - val_loss: 0.0686 - val_a
ccuracy: 0.9830
```

Out[104]: `<tf_keras.src.callbacks.History at 0x28d4bc0d0>`

In [105]:
```
print("QAT Model clusters:")
print_model_weight_clusters(qat_model)
print("\nQAT Model sparsity:")
print_model_weights_sparsity(qat_model)
print("\nPCQAT Model clusters:")
print_model_weight_clusters(pcqat_model)
```

```
print("\nPCQAT Model sparsity:")
print_model_weights_sparsity(pcqat_model)
```

```
QAT Model clusters:
quant_conv2d_6/conv2d_6/kernel:0: 91 clusters
quant_dense_6/dense_6/kernel:0: 16637 clusters

QAT Model sparsity:
conv2d_6/kernel:0: 16.67% sparsity  (18/108)
dense_6/kernel:0: 11.44% sparsity  (2321/20280)

PCQAT Model clusters:
quant_conv2d_6/conv2d_6/kernel:0: 8 clusters
quant_dense_6/dense_6/kernel:0: 8 clusters

PCQAT Model sparsity:
conv2d_6/kernel:0: 53.70% sparsity  (58/108)
dense_6/kernel:0: 56.19% sparsity  (11395/20280)
```

See compression benefits of PCQAT model

Define helper function to get zipped model file.

In [ ]:

Observe that applying sparsity, clustering and PCQAT to a model yields significant compression benefits.

In [106]:
```
# QAT model
converter = tf.lite.TFLiteConverter.from_keras_model(qat_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
qat_tflite_model = converter.convert()
qat_model_file = 'qat_model.tflite'
# Save the model.
with open(qat_model_file, 'wb') as f:
    f.write(qat_tflite_model)


def get_gzipped_model_size2(file):

  _, zipped_file = tempfile.mkstemp('.zip')
  with zipfile.ZipFile(zipped_file, 'w', compression=zipfile.ZIP_DEFLATED) as f:
    f.write(file)

  return os.path.getsize(zipped_file)/1000

# PCQAT model
converter = tf.lite.TFLiteConverter.from_keras_model(pcqat_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
pcqat_tflite_model = converter.convert()
pcqat_model_file = 'pcqat_model.tflite'
# Save the model.
with open(pcqat_model_file, 'wb') as f:
    f.write(pcqat_tflite_model)

# print("QAT model size: ", get_gzipped_model_size(qat_model_file), ' KB')
print("PCQAT model size: ", get_gzipped_model_size2(pcqat_model_file), ' KB')
```

```
INFO:tensorflow:Assets written to: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmprzefcd7b/assets
```

```
INFO:tensorflow:Assets written to: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmprzefcd7b/assets
/Users/ajaymaheshwari/anaconda3/lib/python3.11/site-packages/tensorflow/lite/python/convert.py:964: UserWarning: St
atistics for quantized inputs were expected, but not specified; continuing anyway.
  warnings.warn(
W0000 00:00:1717039150.767356  275574 tf_tfl_flatbuffer_helpers.cc:390] Ignored output_format.
W0000 00:00:1717039150.767863  275574 tf_tfl_flatbuffer_helpers.cc:393] Ignored drop_control_dependency.
2024-05-30 08:49:10.768572: I tensorflow/cc/saved_model/reader.cc:83] Reading SavedModel from: /var/folders/px/z8lb
6znd6q95tq6vlznyb0s40000gn/T/tmprzefcd7b
2024-05-30 08:49:10.769601: I tensorflow/cc/saved_model/reader.cc:51] Reading meta graph with tags { serve }
2024-05-30 08:49:10.769606: I tensorflow/cc/saved_model/reader.cc:146] Reading SavedModel debug info (if present) f
rom: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmprzefcd7b
2024-05-30 08:49:10.778555: I tensorflow/cc/saved_model/loader.cc:234] Restoring SavedModel bundle.
2024-05-30 08:49:10.808503: I tensorflow/cc/saved_model/loader.cc:218] Running initialization op on SavedModel bund
le at path: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmprzefcd7b
2024-05-30 08:49:10.816091: I tensorflow/cc/saved_model/loader.cc:317] SavedModel load for tags { serve }; Status:
success: OK. Took 47521 microseconds.
```

```
INFO:tensorflow:Assets written to: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmp0bbha3yi/assets
```

```
INFO:tensorflow:Assets written to: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmp0bbha3yi/assets
/Users/ajaymaheshwari/anaconda3/lib/python3.11/site-packages/tensorflow/lite/python/convert.py:964: UserWarning: St
atistics for quantized inputs were expected, but not specified; continuing anyway.
  warnings.warn(
```

```
PCQAT model size:  8.266  KB
```

```
W0000 00:00:1717039151.590465  275574 tf_tfl_flatbuffer_helpers.cc:390] Ignored output_format.
W0000 00:00:1717039151.590475  275574 tf_tfl_flatbuffer_helpers.cc:393] Ignored drop_control_dependency.
2024-05-30 08:49:11.590572: I tensorflow/cc/saved_model/reader.cc:83] Reading SavedModel from: /var/folders/px/z8lb
6znd6q95tq6vlznyb0s40000gn/T/tmp0bbha3yi
2024-05-30 08:49:11.591816: I tensorflow/cc/saved_model/reader.cc:51] Reading meta graph with tags { serve }
2024-05-30 08:49:11.591824: I tensorflow/cc/saved_model/reader.cc:146] Reading SavedModel debug info (if present) f
rom: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmp0bbha3yi
2024-05-30 08:49:11.601670: I tensorflow/cc/saved_model/loader.cc:234] Restoring SavedModel bundle.
2024-05-30 08:49:11.634096: I tensorflow/cc/saved_model/loader.cc:218] Running initialization op on SavedModel bund
le at path: /var/folders/px/z8lb6znd6q95tq6vlznyb0s40000gn/T/tmp0bbha3yi
2024-05-30 08:49:11.643307: I tensorflow/cc/saved_model/loader.cc:317] SavedModel load for tags { serve }; Status:
success: OK. Took 52735 microseconds.
```

See the persistence of accuracy from TF to TFLite

Define a helper function to evaluate the TFLite model on the test dataset.

Define a helper function to evaluate the TFLite model on the test dataset.

In [107]:
```
def eval_model(interpreter):
  input_index = interpreter.get_input_details()[0]["index"]
  output_index = interpreter.get_output_details()[0]["index"]

  # Run predictions on every image in the "test" dataset.
  prediction_digits = []
  for i, test_image in enumerate(test_images):
    if i % 1000 == 0:
      print(f"Evaluated on {i} results so far.")
    # Pre-processing: add batch dimension and convert to float32 to match with
    # the model's input data format.
    test_image = np.expand_dims(test_image, axis=0).astype(np.float32)
    interpreter.set_tensor(input_index, test_image)

    # Run inference.
    interpreter.invoke()

    # Post-processing: remove batch dimension and find the digit with highest
    # probability.
    output = interpreter.tensor(output_index)
    digit = np.argmax(output()[0])
    prediction_digits.append(digit)

  print('\n')
  # Compare prediction results with ground truth labels to calculate accuracy.
  prediction_digits = np.array(prediction_digits)
  accuracy = (prediction_digits == test_labels).mean()
  return accuracy
```

Evaluate the model, which has been pruned, clustered and quantized, and then see that the accuracy from TensorFlow persists in the TFLite backend.

In [108]:
```
interpreter = tf.lite.Interpreter(pcqat_model_file)
interpreter.allocate_tensors()

pcqat_test_accuracy = eval_model(interpreter)

print('Pruned, clustered and quantized TFLite test_accuracy:', pcqat_test_accuracy)
print('Baseline TF test accuracy:', baseline_model_accuracy)
```

```
Evaluated on 0 results so far.
Evaluated on 1000 results so far.
Evaluated on 2000 results so far.
Evaluated on 3000 results so far.
Evaluated on 4000 results so far.
Evaluated on 5000 results so far.
Evaluated on 6000 results so far.
Evaluated on 7000 results so far.
Evaluated on 8000 results so far.
Evaluated on 9000 results so far.


Pruned, clustered and quantized TFLite test_accuracy: 0.9818
Baseline TF test accuracy: 0.9830999970436096
```

Conclusion

In this tutorial, you learned how to create a model, prune it using the `prune_low_magnitude()` API, and apply sparsity preserving clustering using the `cluster_weights()` API to preserve sparsity while clustering the weights.

Next, sparsity and cluster preserving quantization aware training (PCQAT) was applied to preserve model sparsity and clusters while using QAT. The final PCQAT model was compared to the QAT one to show that sparsity and clusters are preserved in the former and lost in the latter.

Next, the models were converted to TFLite to show the compression benefits of chaining sparsity, clustering, and PCQAT model optimization techniques and the TFLite model was evaluated to ensure that the accuracy persists in the TFLite backend.

Finally, the PCQAT TFLite model accuracy was compared to the pre-optimization baseline model accuracy to show that collaborative optimization techniques managed to achieve the compression benefits while maintaining a similar accuracy compared to the original model.