```
import tensorflow as tf
import tensorflow_model_optimization as tfmot
import numpy as np
import os
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist

# Load and preprocess MNIST
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = np.stack([train_images] * 3, axis=-1)  # Convert to 3-channel
test_images = np.stack([test_images] * 3, axis=-1)
train_images = tf.image.resize(train_images, [96, 96]) / 255.0
test_images = tf.image.resize(test_images, [96, 96]) / 255.0

# ------------------ BASE MODEL ------------------
def build_transfer_model():
    base_model = tf.keras.applications.MobileNetV2(input_shape=(96, 96, 3),
                                                    include_top=False,
                                                    weights='imagenet')
    base_model.trainable = False

    model = tf.keras.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    return model

base_model = build_transfer_model()
base_model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

base_model.fit(train_images, train_labels,
               epochs=5, batch_size=32, validation_split=0.1, verbose=1)

base_loss, base_acc = base_model.evaluate(test_images, test_labels, verbose=0)
print(f'Base Model Accuracy: {base_acc * 100:.2f}%')

base_model.save('base_model_transfer.keras')
print(f"Base model size: {os.path.getsize('base_model_transfer.keras') / (1024 * 1024):.2f} MB")

# ------------------ PRUNING ------------------
batch_size = 32
epochs = 5
validation_split = 0.1
steps_per_epoch = int(np.ceil((len(train_images) * (1 - validation_split)) / batch_size))
end_step = steps_per_epoch * epochs

pruning_schedule = tfmot.sparsity.keras.PolynomialDecay(
    initial_sparsity=0.0,
    final_sparsity=0.5,
    begin_step=0,
    end_step=end_step
)

prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

def build_pruned_model():
    base_model = tf.keras.applications.MobileNetV2(input_shape=(96, 96, 3),
                                                    include_top=False,
                                                    weights='imagenet')
    base_model.trainable = False
    model = tf.keras.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    return prune_low_magnitude(model, pruning_schedule=pruning_schedule)

pruned_model = build_pruned_model()
pruned_model.compile(optimizer='adam',
                     loss='sparse_categorical_crossentropy',
```

```
                      metrics=['accuracy'])

callbacks = [
    tfmot.sparsity.keras.UpdatePruningStep(),
    tfmot.sparsity.keras.PruningSummaries(log_dir='/tmp/pruning_logs')
]

pruned_model.fit(train_images, train_labels,
                 batch_size=batch_size,
                 epochs=epochs,
                 validation_split=validation_split,
                 callbacks=callbacks,
                 verbose=1)

# Strip and recompile
stripped_model = tfmot.sparsity.keras.strip_pruning(pruned_model)
stripped_model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

pruned_loss, pruned_acc = stripped_model.evaluate(test_images, test_labels, verbose=0)
print(f'Pruned Model Accuracy: {pruned_acc * 100:.2f}%')

stripped_model.save('pruned_model_transfer.keras')
print(f"Pruned model size: {os.path.getsize('pruned_model_transfer.keras') / (1024 * 1024):.2f} MB")

# ------------------ QUANTIZATION ------------------
converter = tf.lite.TFLiteConverter.from_keras_model(stripped_model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]

quantized_model = converter.convert()
with open('quantized_model_transfer.tflite', 'wb') as f:
    f.write(quantized_model)

quantized_model_size = os.path.getsize('quantized_model_transfer.tflite') / (1024 * 1024)
print(f"Quantized model size: {quantized_model_size:.2f} MB")

# ------------------ EVALUATE TFLITE ------------------
interpreter = tf.lite.Interpreter(model_path="quantized_model_transfer.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

correct = 0
for i in range(len(test_images)):
    input_data = np.expand_dims(test_images[i], axis=0).astype(np.float32)
    interpreter.set_tensor(input_details[0]['index'], input_data)
    interpreter.invoke()
    output = interpreter.get_tensor(output_details[0]['index'])
    if np.argmax(output[0]) == test_labels[i]:
        correct += 1

print(f"Quantized TFLite Model Accuracy: {correct / len(test_images) * 100:.2f}%")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_
9406464/9406464 [==============================] - 0s 0us/step
Epoch 1/5
1688/1688 [==============================] - 446s 263ms/step - loss: 0.1624 - accuracy: 0.9474 - val_loss: 0.0924 - val_accuracy: 0.9695
Epoch 2/5
1688/1688 [==============================] - 455s 270ms/step - loss: 0.0884 - accuracy: 0.9702 - val_loss: 0.0931 - val_accuracy: 0.9687
Epoch 3/5
1688/1688 [==============================] - 449s 266ms/step - loss: 0.0706 - accuracy: 0.9755 - val_loss: 0.0839 - val_accuracy: 0.9715
Epoch 4/5
1688/1688 [==============================] - 445s 264ms/step - loss: 0.0592 - accuracy: 0.9796 - val_loss: 0.0744 - val_accuracy: 0.9766
Epoch 5/5
1688/1688 [==============================] - 446s 264ms/step - loss: 0.0479 - accuracy: 0.9838 - val_loss: 0.0798 - val_accuracy: 0.9757
Base Model Accuracy: 97.25%
Base model size: 10.81 MB
Epoch 1/5
1688/1688 [==============================] - 509s 283ms/step - loss: 0.1831 - accuracy: 0.9405 - val_loss: 0.1356 - val_accuracy: 0.9546
Epoch 2/5
1688/1688 [==============================] - 462s 274ms/step - loss: 0.2340 - accuracy: 0.9231 - val_loss: 0.1759 - val_accuracy: 0.9433
Epoch 3/5
1688/1688 [==============================] - 452s 268ms/step - loss: 0.3907 - accuracy: 0.8744 - val_loss: 0.4007 - val_accuracy: 0.8782
Epoch 4/5
1688/1688 [==============================] - 459s 272ms/step - loss: 0.4617 - accuracy: 0.8528 - val_loss: 0.3654 - val_accuracy: 0.8883
```

```
Epoch 5/5
1688/1688 [==============================] - 460s 273ms/step - loss: 0.3168 - accuracy: 0.9000 - val_loss: 0.2239 - val_accuracy: 0.9288
Pruned Model Accuracy: 91.78%
Pruned model size: 9.54 MB
WARNING:absl:Function `_wrapped_model` contains input name(s) mobilenetv2_1.00_96_input with unsupported characters which will be rename
WARNING:absl:`mobilenetv2_1.00_96_input` is not a valid tf.function parameter name. Sanitizing to `mobilenetv2_1_00_96_input`.
WARNING:absl:`mobilenetv2_1.00_96_input` is not a valid tf.function parameter name. Sanitizing to `mobilenetv2_1_00_96_input`.
WARNING:absl:`mobilenetv2_1.00_96_input` is not a valid tf.function parameter name. Sanitizing to `mobilenetv2_1_00_96_input`.
WARNING:absl:Found untraced functions such as _update_step_xla, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compile
Quantized model size: 4.58 MB
Quantized TFLite Model Accuracy: 92.02%
```

```
import tensorflow as tf
import tensorflow_model_optimization as tfmot
import numpy as np
import os
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist

# Load and preprocess MNIST
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

train_images = np.stack([train_images] * 3, axis=-1)  # Convert to 3-channel
test_images = np.stack([test_images] * 3, axis=-1)
train_images = tf.image.resize(train_images, [96, 96]) / 255.0
test_images = tf.image.resize(test_images, [96, 96]) / 255.0

# ------------------ BASE MODEL ------------------
def build_transfer_model():
    base_model = tf.keras.applications.MobileNetV2(input_shape=(96, 96, 3),
                                                    include_top=False,
                                                    weights='imagenet')
    base_model.trainable = False

    model = tf.keras.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    return model

base_model = build_transfer_model()
base_model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

base_model.fit(train_images, train_labels,
               epochs=5, batch_size=32, validation_split=0.1, verbose=1)

base_loss, base_acc = base_model.evaluate(test_images, test_labels, verbose=0)
print(f'Base Model Accuracy: {base_acc * 100:.2f}%')

base_model.save('base_model_transfer.keras')
print(f"Base model size: {os.path.getsize('base_model_transfer.keras') / (1024 * 1024):.2f} MB")

# ------------------ PRUNING ------------------
batch_size = 32
epochs = 5
validation_split = 0.1
steps_per_epoch = int(np.ceil((len(train_images) * (1 - validation_split)) / batch_size))
end_step = steps_per_epoch * epochs

pruning_schedule = tfmot.sparsity.keras.PolynomialDecay(
    initial_sparsity=0.0,
    final_sparsity=0.5,
    begin_step=0,
    end_step=end_step
)

prune_low_magnitude = tfmot.sparsity.keras.prune_low_magnitude

def build_pruned_model():
    base_model = tf.keras.applications.MobileNetV2(input_shape=(96, 96, 3),
                                                    include_top=False,
                                                    weights='imagenet')
```

```
        base_model.trainable = False
        model = tf.keras.Sequential([
            base_model,
            layers.GlobalAveragePooling2D(),
            layers.Dense(128, activation='relu'),
            layers.Dense(10, activation='softmax')
        ])
        return prune_low_magnitude(model, pruning_schedule=pruning_schedule)

    pruned_model = build_pruned_model()
    pruned_model.compile(optimizer='adam',
                         loss='sparse_categorical_crossentropy',
                         metrics=['accuracy'])

    callbacks = [
        tfmot.sparsity.keras.UpdatePruningStep(),
        tfmot.sparsity.keras.PruningSummaries(log_dir='/tmp/pruning_logs')
    ]

    pruned_model.fit(train_images, train_labels,
                     batch_size=batch_size,
                     epochs=epochs,
                     validation_split=validation_split,
                     callbacks=callbacks,
                     verbose=1)

    # Strip and recompile
    stripped_model = tfmot.sparsity.keras.strip_pruning(pruned_model)
    stripped_model.compile(optimizer='adam',
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])

    pruned_loss, pruned_acc = stripped_model.evaluate(test_images, test_labels, verbose=0)
    print(f'Pruned Model Accuracy: {pruned_acc * 100:.2f}%')

    stripped_model.save('pruned_model_transfer.keras')
    print(f"Pruned model size: {os.path.getsize('pruned_model_transfer.keras') / (1024 * 1024):.2f} MB")

    # ------------------ QUANTIZATION ------------------
    converter = tf.lite.TFLiteConverter.from_keras_model(stripped_model)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    converter.target_spec.supported_types = [tf.float16]

    quantized_model = converter.convert()
    with open('quantized_model_transfer.tflite', 'wb') as f:
        f.write(quantized_model)

    quantized_model_size = os.path.getsize('quantized_model_transfer.tflite') / (1024 * 1024)
    print(f"Quantized model size: {quantized_model_size:.2f} MB")

    # ------------------ EVALUATE TFLITE ------------------
    interpreter = tf.lite.Interpreter(model_path="quantized_model_transfer.tflite")
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    correct = 0
    for i in range(len(test_images)):
        input_data = np.expand_dims(test_images[i], axis=0).astype(np.float32)
        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details[0]['index'])
        if np.argmax(output[0]) == test_labels[i]:
            correct += 1

    print(f"Quantized TFLite Model Accuracy: {correct / len(test_images) * 100:.2f}%")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_
9406464/9406464 [==============================] - 0s 0us/step
Epoch 1/5
```

```
1688/1688 [==============================] - 491s 289ms/step - loss: 0.3689 - accuracy: 0.8657 - val_loss: 0.3069 - val_accuracy: 0.8877
Epoch 2/5
1688/1688 [==============================] - 462s 274ms/step - loss: 0.2796 - accuracy: 0.8959 - val_loss: 0.2886 - val_accuracy: 0.8918
Epoch 3/5
1688/1688 [==============================] - 467s 277ms/step - loss: 0.2445 - accuracy: 0.9091 - val_loss: 0.3045 - val_accuracy: 0.8885
Epoch 4/5
1688/1688 [==============================] - 459s 272ms/step - loss: 0.2239 - accuracy: 0.9160 - val_loss: 0.2681 - val_accuracy: 0.9035
Epoch 5/5
1688/1688 [==============================] - 459s 272ms/step - loss: 0.2011 - accuracy: 0.9236 - val_loss: 0.2866 - val_accuracy: 0.8983
Base Model Accuracy: 89.64%
Base model size: 10.81 MB
Epoch 1/5
1688/1688 [==============================] - 534s 298ms/step - loss: 0.3976 - accuracy: 0.8561 - val_loss: 0.3796 - val_accuracy: 0.8575
Epoch 2/5
1688/1688 [==============================] - 489s 290ms/step - loss: 0.4925 - accuracy: 0.8215 - val_loss: 0.5427 - val_accuracy: 0.8027
Epoch 3/5
1688/1688 [==============================] - 480s 284ms/step - loss: 0.6512 - accuracy: 0.7671 - val_loss: 0.6988 - val_accuracy: 0.7583
Epoch 4/5
1688/1688 [==============================] - 491s 291ms/step - loss: 0.7154 - accuracy: 0.7431 - val_loss: 0.6475 - val_accuracy: 0.7726
Epoch 5/5
1688/1688 [==============================] - 489s 290ms/step - loss: 0.5885 - accuracy: 0.7886 - val_loss: 0.5382 - val_accuracy: 0.8102
Pruned Model Accuracy: 80.96%
Pruned model size: 9.54 MB
WARNING:absl:Function `_wrapped_model` contains input name(s) mobilenetv2_1.00_96_input with unsupported characters which will be rename
WARNING:absl:`mobilenetv2_1.00_96_input` is not a valid tf.function parameter name. Sanitizing to `mobilenetv2_1_00_96_input`.
WARNING:absl:`mobilenetv2_1.00_96_input` is not a valid tf.function parameter name. Sanitizing to `mobilenetv2_1_00_96_input`.
WARNING:absl:`mobilenetv2_1.00_96_input` is not a valid tf.function parameter name. Sanitizing to `mobilenetv2_1_00_96_input`.
WARNING:absl:Found untraced functions such as _update_step_xla, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compile
Quantized model size: 4.58 MB
Quantized TFLite Model Accuracy: 81.02%
```