Model3_FT - FineTuning using ConvNext

by Kaushik Srivatsan - CDS - kaushik.s-25@scds.saiuniversity.edu.in

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import tensorflow as tf
print(tf.__version__)

from tensorflow import keras
tf.random.set_seed(42)

import numpy as np
np.random.seed(42)

import matplotlib.pyplot as plt
%matplotlib inline

import glob
import PIL
from PIL import Image
```

```
2.15.0
```

Loading the preprocessed dataset

```
# load numpy array from npy file
from numpy import load

X_train_std = load('/content/drive/MyDrive/Models/X_train_std_model3.npy')
X_test_std = load('/content/drive/MyDrive/Models/X_test_std_model3.npy')

y_train = load('/content/drive/MyDrive/Models/y_train_model3.npy')
y_test = load('/content/drive/MyDrive/Models/y_test_model3.npy')
```

```
print("X_train_std_shape: {}".format(X_train_std.shape))
print("X_test_std_shape: {}".format(X_test_std.shape))
```

```
X_train_std_shape: (373, 299, 299, 3)
X_test_std_shape: (125, 299, 299, 3)
```

Loading the Transfer-learning Model

```
#Importing layers and stochastic depth to mitigate the export issue of keras with respect to the convnext model
from keras import layers

class StochasticDepth(layers.Layer):
    """Stochastic Depth module.

    It performs batch-wise dropping rather than sample-wise. In libraries like
    `timm`, it's similar to `DropPath` layers that drops residual paths
    sample-wise.

    References:
      - https://github.com/rwightman/pytorch-image-models

    Args:
      drop_path_rate (float): Probability of dropping paths. Should be within
        [0, 1].

    Returns:
      Tensor either with the residual path dropped or kept.
    """

    def __init__(self, drop_path_rate, **kwargs):
        super().__init__(**kwargs)
        self.drop_path_rate = drop_path_rate

    def call(self, x, training=None):
        if training:
            keep_prob = 1 - self.drop_path_rate
            shape = (tf.shape(x)[0],) + (1,) * (len(tf.shape(x)) - 1)
            random_tensor = keep_prob + tf.random.uniform(shape, 0, 1)
            random_tensor = tf.floor(random_tensor)
            return (x / keep_prob) * random_tensor
        return x

    def get_config(self):
        config = super().get_config()
        config.update({"drop_path_rate": self.drop_path_rate})
        return config

class LayerScale(layers.Layer):
    """Layer scale module.

    References:
      - https://arxiv.org/abs/2103.17239

    Args:
      init_values (float): Initial value for layer scale. Should be within
        [0, 1].
      projection_dim (int): Projection dimensionality.

    Returns:
      Tensor multiplied to the scale.
    """

    def __init__(self, init_values, projection_dim, **kwargs):
        super().__init__(**kwargs)
        self.init_values = init_values
        self.projection_dim = projection_dim

    def build(self, input_shape):
        self.gamma = tf.Variable(
            self.init_values * tf.ones((self.projection_dim,))
        )

    def call(self, x):
        return x * self.gamma

    def get_config(self):
        config = super().get_config()
        config.update(
            {
                "init_values": self.init_values,
                "projection_dim": self.projection_dim,
            }
        )
        return config

model3_FT = keras.models.load_model('/content/drive/MyDrive/Models/Model3_TL.h5',  compile=False, custom_objects={ "LayerScale": LayerSca
model3_FT.summary()
```

```
 e)

 convnext_tiny_stage_3_bloc  (None, 9, 9, 768)            0         ['convnext_tiny_stage_3_block_
 k_2_identity (Activation)                                          2_layer_scale[0][0]']

 tf.__operators__.add_17 (T  (None, 9, 9, 768)            0         ['tf.__operators__.add_16[0][0
 FOpLambda)                                                         ]',
                                                                     'convnext_tiny_stage_3_block_
                                                                    2_identity[0][0]']

 layer_normalization (Layer  (None, 9, 9, 768)            1536      ['tf.__operators__.add_17[0][0
 Normalization)                                                     ]']

 global_average_pooling2d (  (None, 768)                  0         ['layer_normalization[0][0]']
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 768)                  0         ['global_average_pooling2d[0][
                                                                    0]']

 dense (Dense)               (None, 4)                    3076      ['dropout[0][0]']

==================================================================================================
Total params: 27823204 (106.14 MB)
Trainable params: 3076 (12.02 KB)
Non-trainable params: 27820128 (106.13 MB)
__________________________________________________________________________________________________
```

Modifyng and Fine tuning the layers to be trained

```
total_layers = len(model3_FT.layers)
split_index = int(0.25 * total_layers)

for layer in model3_FT.layers[:split_index]:
    layer.trainable = False

for layer in model3_FT.layers[split_index:]:
    layer.trainable = True
```

Compiling and Training the Model

```
model3_FT.compile(loss='sparse_categorical_crossentropy',
                 optimizer='adam',
                 metrics=['accuracy'])

callbacks_FineTune = [
             keras.callbacks.ModelCheckpoint("bestFT1.h5",
                                             monitor='val_accuracy',
                                             save_weights_only=True,
                                             save_best_only=True)
]

history_FineTune = model3_FT.fit(x = X_train_std, y = y_train, epochs=10,
                                      validation_split=0.1, batch_size=16, callbacks=callbacks_FineTune)
```

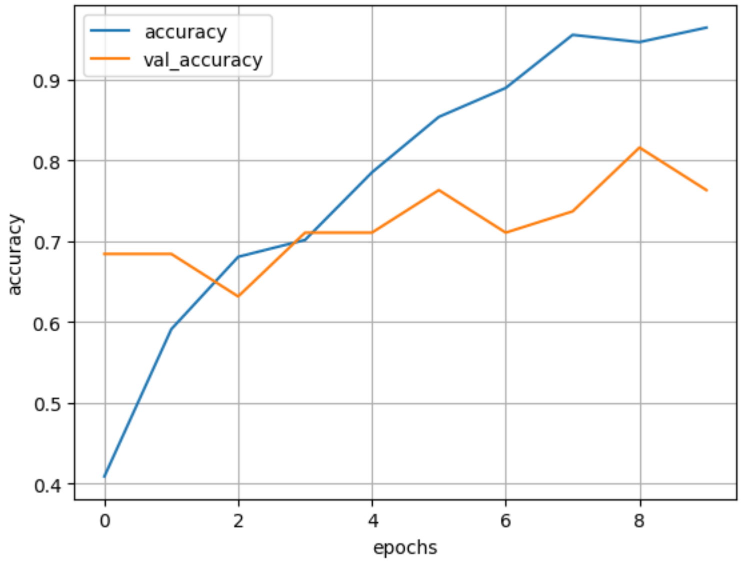
```
Epoch 1/10
21/21 [==============================] - 66s 1s/step - loss: 1.4309 - accuracy: 0.4090 - val_loss: 0.9756 - val_accuracy: 0.6842
Epoch 2/10
21/21 [==============================] - 9s 424ms/step - loss: 0.9142 - accuracy: 0.5910 - val_loss: 0.7564 - val_accuracy: 0.6842
Epoch 3/10
21/21 [==============================] - 9s 427ms/step - loss: 0.7681 - accuracy: 0.6806 - val_loss: 0.7064 - val_accuracy: 0.6316
Epoch 4/10
21/21 [==============================] - 9s 443ms/step - loss: 0.6766 - accuracy: 0.7015 - val_loss: 0.7243 - val_accuracy: 0.7105
Epoch 5/10
21/21 [==============================] - 9s 425ms/step - loss: 0.5328 - accuracy: 0.7851 - val_loss: 0.6848 - val_accuracy: 0.7105
Epoch 6/10
21/21 [==============================] - 9s 450ms/step - loss: 0.3814 - accuracy: 0.8537 - val_loss: 0.5090 - val_accuracy: 0.7632
Epoch 7/10
21/21 [==============================] - 9s 425ms/step - loss: 0.3014 - accuracy: 0.8896 - val_loss: 0.6094 - val_accuracy: 0.7105
Epoch 8/10
21/21 [==============================] - 9s 434ms/step - loss: 0.1725 - accuracy: 0.9552 - val_loss: 1.0989 - val_accuracy: 0.7368
Epoch 9/10
21/21 [==============================] - 9s 453ms/step - loss: 0.1464 - accuracy: 0.9463 - val_loss: 0.6187 - val_accuracy: 0.8158
Epoch 10/10
21/21 [==============================] - 9s 422ms/step - loss: 0.1158 - accuracy: 0.9642 - val_loss: 0.7829 - val_accuracy: 0.7632
```

```
keys = ['accuracy', 'val_accuracy']
progress = {k:v for k,v in history_FineTune.history.items() if k in keys}

import pandas as pd
pd.DataFrame(progress).plot()

plt.xlabel("epochs")
plt.ylabel("accuracy")

plt.grid(True)
plt.show()
```



Evaluating the Model with Best weights

```
model3_FT.load_weights("bestFT1.h5")

testLoss_FineTune, testAccuracy_FineTune = model3_FT.evaluate(x = X_test_std, y = y_test)

print("Test-loss: %f, Test-accuracy: %f" % (testLoss_FineTune, testAccuracy_FineTune))
```

```
4/4 [==============================] - 8s 1s/step - loss: 1.2114 - accuracy: 0.7360
Test-loss: 1.211351, Test-accuracy: 0.736000
```

```
## Checking Model performance
```

```
y_proba = model3_FT.predict(X_test_std)
y_predict = np.argmax(y_proba, axis=-1)
print(y_predict)
```

```
4/4 [==============================] - 6s 404ms/step
[3 2 1 3 0 3 3 0 1 0 3 2 3 0 1 1 1 3 1 1 1 2 0 2 1 0 3 2 3 3 3 3 0 0 1 0 2
 3 2 1 3 3 2 0 1 0 1 3 0 3 1 0 0 0 0 0 2 1 2 2 0 3 3 3 3 3 2 1 1 1 2 3 2 3
 3 1 3 0 0 3 2 0 0 2 2 0 3 3 0 0 1 2 2 3 3 2 2 0 1 0 1 0 3 1 0 3 3 1 0 3 3
 2 0 2 3 3 0 3 1 0 3 3 3 3 0]
```

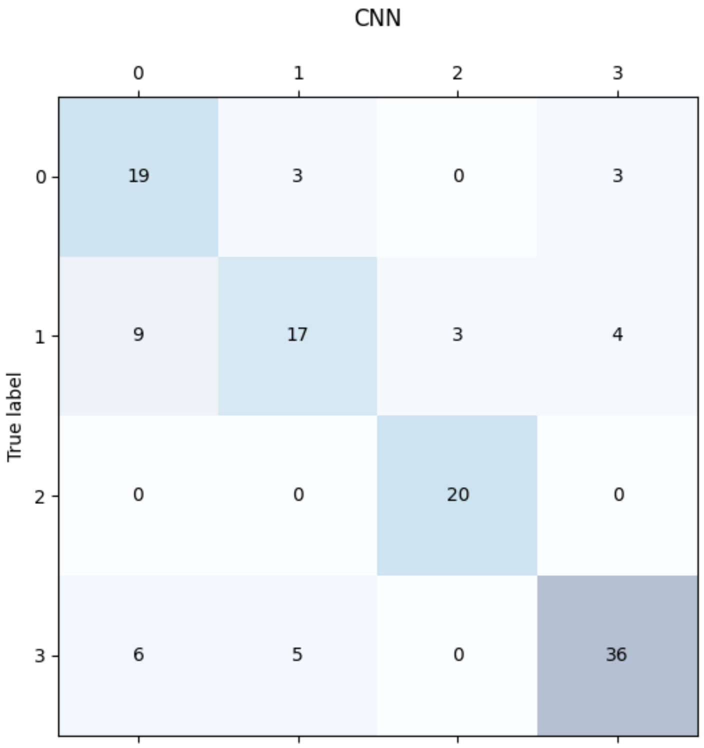
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true = y_test, y_pred = y_predict)

fig, ax = plt.subplots(figsize=(6, 6))
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(x=j, y=i, s=cm[i, j], va='center', ha='center')

ax.title.set_text('CNN\n')
plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.tight_layout()
plt.savefig("ConfusionMatrix.png", dpi=300, format='png', pad_inches=0.3)
plt.show()
```



	0	1	2	3
0	19	3	0	3
1	9	17	3	4
2	0	0	20	0
3	6	5	0	36

```
from sklearn.metrics import precision_score, recall_score, f1_score

pScore = precision_score(y_true= y_test, y_pred = y_predict, average = 'weighted')
print("Precision: ", pScore)

rScore = recall_score(y_true= y_test, y_pred = y_predict, average = 'weighted')
print("Recall: ", rScore)

fScore = f1_score(y_true= y_test, y_pred = y_predict, average = 'weighted')
print("F1-score: ", fScore)
```

```
Precision:  0.7452058383393803
Recall:  0.736
F1-score:  0.7332093893140147
```

Saving the Fine-Tuned Model

```
model3_FT.save('/content/drive/MyDrive/Models/model3_FT.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
```