```
from google.colab import drive

from tensorflow.keras import layers, Sequential, Input, Model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing import image_dataset_from_directory

import numpy as np
import matplotlib.pyplot as plt

from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
from torchvision.transforms import Resize, Normalize, ToTensor, Compose, RandomVerticalFlip, RandomHorizontalFlip, RandomPerspective, RandomInvert, Ran

drive.mount('/content/drive')
dataset_path = '/content/drive/MyDrive/fake_image_detection'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
def get_dataset_split_tensorflow(directory, image_size=(256,256), batch_size=32):
  train_data  = image_dataset_from_directory(directory=f'{dataset_path}/train', label_mode='binary', image_size=image_size, batch_size=batch_size)
  val_data    = image_dataset_from_directory(directory=f'{dataset_path}/valid', label_mode='binary', image_size=image_size, batch_size=batch_size)
  test_data   = image_dataset_from_directory(directory=f'{dataset_path}/test', label_mode='binary', image_size=image_size, batch_size=batch_size)

  return train_data, val_data, test_data
```

```
train_data_tensorflow, val_data_tensorflow, test_data_tensorflow = get_dataset_split_tensorflow(dataset_path)
```

```
Found 189 files belonging to 2 classes.
Found 21 files belonging to 2 classes.
Found 90 files belonging to 2 classes.
```

```
data_augmentation = Sequential([layers.RandomFlip("horizontal_and_vertical"), layers.RandomTranslation(0.3,0.3), layers.RandomRotation(0.4), layers.Ran
```

```
from keras import backend as K

def get_recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def get_precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def get_f1(y_true, y_pred):
    precision = get_precision(y_true, y_pred)
    recall = get_recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

```
from keras import initializers

def CNNmodel(input_shape):
    inputs = Input(shape=input_shape+(3,))
    x = data_augmentation(inputs)

    # 1st Convolutional Layer
    x = layers.Conv2D(16, 3, strides = 2, kernel_initializer = 'he_uniform',
    bias_initializer = initializers.Zeros(), padding = 'same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(3, strides = 1, padding = 'same')(x)

    # 2nd Convolutional Layer
    x = layers.Conv2D(32, 3, strides = 2, kernel_initializer = 'he_uniform',
    bias_initializer = initializers.Zeros(), padding = 'same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(3, strides = 1, padding = 'same')(x)

    # 3rd Convolutional Layer
    x = layers.Conv2D(64, 3, strides = 2, kernel_initializer = 'he_uniform',
    bias_initializer = initializers.Zeros(), padding = 'same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(3, strides = 1, padding = 'same')(x)

    # 4th Convolutional Layer
    x = layers.Conv2D(64, 3, strides = 2, kernel_initializer = 'he_uniform',
    bias_initializer = initializers.Zeros(), padding = 'same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(3, strides = 1, padding='same')(x)
```

```
    # Passing it to a Fully Connected layer
    x = layers.Flatten()(x)

    # 1st Fully Connected Layer
    x = layers.Dense(256, kernel_initializer = 'he_uniform',
    bias_initializer=initializers.Zeros())(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    # Output Layer
    outputs = layers.Dense(1, activation="sigmoid")(x)
    return Model(inputs, outputs)


model = CNNmodel(input_shape=(256,256))
model.summary()
```

```
 sequential_2 (Sequential)   (None, 256, 256, 3)       0

 conv2d_60 (Conv2D)          (None, 128, 128, 16)      448

 batch_normalization_75 (Bat  (None, 128, 128, 16)     64
 chNormalization)

 activation_75 (Activation)  (None, 128, 128, 16)      0

 max_pooling2d_60 (MaxPoolin  (None, 128, 128, 16)     0
 g2D)

 conv2d_61 (Conv2D)          (None, 64, 64, 32)        4640

 batch_normalization_76 (Bat  (None, 64, 64, 32)       128
 chNormalization)

 activation_76 (Activation)  (None, 64, 64, 32)        0

 max_pooling2d_61 (MaxPoolin  (None, 64, 64, 32)       0
 g2D)

 conv2d_62 (Conv2D)          (None, 32, 32, 64)        18496

 batch_normalization_77 (Bat  (None, 32, 32, 64)       256
 chNormalization)

 activation_77 (Activation)  (None, 32, 32, 64)        0

 max_pooling2d_62 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 conv2d_63 (Conv2D)          (None, 16, 16, 64)        36928

 batch_normalization_78 (Bat  (None, 16, 16, 64)       256
 chNormalization)

 activation_78 (Activation)  (None, 16, 16, 64)        0

 max_pooling2d_63 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 flatten_17 (Flatten)        (None, 16384)             0

 dense_38 (Dense)            (None, 256)               4194560

 batch_normalization_79 (Bat  (None, 256)              1024
 chNormalization)

 activation_79 (Activation)  (None, 256)               0

 dense_39 (Dense)            (None, 1)                 257

=================================================================
Total params: 4,257,057
Trainable params: 4,256,193
Non-trainable params: 864
_________________________________________________________________
```

```
from tensorflow.keras import optimizers

epochs = 1
optimizer = optimizers.Adam()
loss = "binary_crossentropy"

model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy',get_f1, get_precision, get_recall])
history = model.fit(train_data_tensorflow, epochs=epochs, validation_data=val_data_tensorflow)
```

```
6/6 [==============================] - 25s 3s/step - loss: 1.6829 - accuracy: 0.4339 - get_f1: 0.4722 - get_precision: 0.4567 - get_recall: 0.518
```

```
test_loss, test_acc, test_f1, test_precision, test_recall = model.evaluate(test_data_tensorflow)
print('\nTest accuracy:', round(test_acc*100,4))
print('Test recall:', round(test_recall*100,4))
print('Test f1_score:', round(test_f1*100,4))
```

```
3/3 [==============================] - 1s 226ms/step - loss: 4.8495 - accuracy: 0.5222 - get_f1: 0.6560 - get_precision: 0.5035 - get_recall: 0.9
```

```
Test accuracy: 52.2222
Test recall: 98.4127
Test f1_score: 65.6022
```

2. Tuning the CNN model

```
from tensorflow.keras import applications, regularizers

def ResNet(pre_trained, input_shape):
  inputs = Input(shape=input_shape+(3,))
  x = data_augmentation(inputs)
  x = pre_trained(x)
  x = layers.Flatten()(x)

  x = layers.Dropout(0.5)(x)
  x = layers.Dense(1024, activation="relu")(x)
  x = layers.Dense(512, activation="relu")(x)
  x = layers.Dense(128, activation="relu")(x)
  output = layers.Dense(1, activation='sigmoid')(x)
  return Model(inputs, output)

pre_trained = applications.ResNet50(
    include_top=False,
    weights="imagenet",
    input_shape=(256,256,3),
)

pre_trained.trainable = False

tuned_model = ResNet(pre_trained, input_shape=(256,256))
tuned_model.summary()
```

```
Model: "model_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_25 (InputLayer)       [(None, 256, 256, 3)]     0

 sequential_2 (Sequential)   (None, 256, 256, 3)       0

 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 flatten_18 (Flatten)        (None, 131072)            0

 dropout_2 (Dropout)         (None, 131072)            0

 dense_40 (Dense)            (None, 1024)              134218752

 dense_41 (Dense)            (None, 512)               524800

 dense_42 (Dense)            (None, 128)               65664

 dense_43 (Dense)            (None, 1)                 129

=================================================================
Total params: 158,397,057
Trainable params: 134,809,345
Non-trainable params: 23,587,712
_________________________________________________________________
```

```
from tensorflow.keras import optimizers
from tensorflow.keras import losses

epochs = 10
optimizer = optimizers.Adam(learning_rate=0.01)
loss = losses.BinaryCrossentropy(from_logits=False)

tuned_model.compile(optimizer=optimizer, loss=loss, metrics=['acc', get_f1, get_precision, get_recall])
history = tuned_model.fit(train_data_tensorflow, epochs=epochs, validation_data=val_data_tensorflow)
```

```
Epoch 1/10
6/6 [==============================] - 78s 12s/step - loss: 1759.3196 - acc: 0.4233 - get_f1: 0.3604 - get_precision: 0.3427 - get_recall: 0.5577
Epoch 2/10
6/6 [==============================] - 72s 12s/step - loss: 5.5496 - acc: 0.4392 - get_f1: 0.4108 - get_precision: 0.3055 - get_recall: 0.6667 -
Epoch 3/10
6/6 [==============================] - 77s 13s/step - loss: 4.7985 - acc: 0.5079 - get_f1: 0.3371 - get_precision: 0.2552 - get_recall: 0.5000 -
Epoch 4/10
6/6 [==============================] - 69s 12s/step - loss: 4.7167 - acc: 0.6085 - get_f1: 0.3764 - get_precision: 0.3044 - get_recall: 0.5000 -
Epoch 5/10
6/6 [==============================] - 69s 12s/step - loss: 3.0805 - acc: 0.4180 - get_f1: 0.2943 - get_precision: 0.2091 - get_recall: 0.5000 -
Epoch 6/10
6/6 [==============================] - 73s 12s/step - loss: 1.3664 - acc: 0.4921 - get_f1: 0.3306 - get_precision: 0.2471 - get_recall: 0.5000 -
Epoch 7/10
6/6 [==============================] - 75s 13s/step - loss: 0.9957 - acc: 0.4709 - get_f1: 0.2099 - get_precision: 0.1539 - get_recall: 0.3333 -
Epoch 8/10
6/6 [==============================] - 69s 12s/step - loss: 0.9109 - acc: 0.4868 - get_f1: 0.3266 - get_precision: 0.2448 - get_recall: 0.5000 -
Epoch 9/10
6/6 [==============================] - 73s 12s/step - loss: 0.7870 - acc: 0.5079 - get_f1: 0.4481 - get_precision: 0.3385 - get_recall: 0.6667 -
Epoch 10/10
6/6 [==============================] - 73s 12s/step - loss: 0.7720 - acc: 0.4815 - get_f1: 0.2117 - get_precision: 0.1570 - get_recall: 0.3333 -
```

```
test_loss_tuned, test_acc_tuned, test_f1_tuned, test_precision_tuned, test_recall_tuned = tuned_model.evaluate(test_data_tensorflow)
print('\nTest accuracy:', round(test_acc_tuned*100,4))
print('Test recall:', round(test_recall_tuned*100,4))
print('Test f1_score:', round(test_f1_tuned*100,4))
```

```
3/3 [==============================] - 26s 9s/step - loss: 0.7693 - acc: 0.5000 - get_f1: 0.6651 - get_precision: 0.5000 - get_recall: 1.0000

Test accuracy: 50.0
Test recall: 100.0
Test f1_score: 66.5121
```

```
from keras import optimizers, regularizers
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    verbose=1,
    mode='min',
    restore_best_weights=True
)

learning_rate = 0.0001
batch_size = 32
num_epoch = 5
optimizers = [optimizers.SGD(), optimizers.Adam()]
regularizers = [regularizers.l1(0.01), regularizers.l2(0.01), regularizers.l1_l2(l1=0.01, l2=0.01)]


for opt in optimizers:
    for reg in regularizers:
        model = CNNmodel(input_shape = (256, 256))
        model.compile(
            optimizer = opt,
            loss = 'binary_crossentropy',
            metrics = ['accuracy']
        )

        history = model.fit(
            train_data_tensorflow,
            epochs = num_epoch,
            batch_size = batch_size,
            validation_data = val_data_tensorflow,
            callbacks = [early_stopping],
            verbose = 1
        )
        # Evaluate the model
        loss, acc = model.evaluate(test_data_tensorflow)
        print(f'\nOptimizer: {opt}, \nRegularizer: {reg}, \n loss: {loss}, \n accuracy: {acc}\n')
```

```
Epoch 1/5
6/6 [==============================] - 15s 2s/step - loss: 1.6891 - accuracy: 0.4921 - val_loss: 7.5799 - val_accuracy: 0.5238
Epoch 2/5
6/6 [==============================] - 13s 2s/step - loss: 0.9084 - accuracy: 0.4709 - val_loss: 2.0537 - val_accuracy: 0.5238
Epoch 3/5
6/6 [==============================] - 13s 2s/step - loss: 0.6636 - accuracy: 0.5661 - val_loss: 7.6989 - val_accuracy: 0.4762
Epoch 4/5
6/6 [==============================] - 18s 3s/step - loss: 0.7550 - accuracy: 0.5397 - val_loss: 8.0562 - val_accuracy: 0.4762
Epoch 5/5
6/6 [==============================] - 13s 2s/step - loss: 0.7145 - accuracy: 0.5503 - val_loss: 5.2279 - val_accuracy: 0.4762
3/3 [==============================] - 1s 226ms/step - loss: 6.0498 - accuracy: 0.4889

Optimizer: <keras.optimizers.legacy.adam.Adam object at 0x7f8c4409d1c0>,
Regularizer: <keras.regularizers.L1L2 object at 0x7f8c4409d250>,
 loss: 6.0497565269470215,
 accuracy: 0.4888888895511627
```

Optimizer: optimizers.Adam()

Regularizer: regularizers.l1_l2(l1=0.01, l2=0.01)

Loss: 1.2607940435409546

Accuracy: 0.5444444417953491