Research proposal

Primary Objectives

Current monitoring methods for bird species require manual labour where specially trained individuals are deployed to often hostile or inaccessible locations to manually count the number of observed species. This places limits on the practicality of the observation methods, often restricting the quality and correctness of the observations. The objective of this work is to develop a reliable and robust computer vision model capable of classifying the species of a bird from an input image. The successful development of such a model could be deployed in the field in conjunction with the use of remote “trail” cameras, allowing for accurate monitoring of species populations without the requirement of manual labour.

Previous Work

Numerous attempts into reducing the manual labour required for species monitoring have been attempted, with some theorized methods opting to instead use audio signals, such as [1] (https://www.sciencedirect.com/science/article/pii/S0167865509002487), [2] (https://ieeexplore.ieee.org/abstract/document/5360230), [3] (https://ieeexplore.ieee.org/abstract/document/6083794) and [4] (https://ieeexplore.ieee.org/abstract/document/5946906) as no line of sight is required for these methods. These attempts found that although viable, the audio processing route was susceptible to numerous disadvantages due to the signals being sparse (a bird may not produce a sound for quite a while) as well as being unable to count the number of birds (is this call from the same bird or a different bird of the same species). As such, other attempts have used computer vision and image processing-based techniques such as [5] (https://openaccess.thecvf.com/content_cvpr_2013/html/Berg_POOF_Part-Based_One-vs.-One_2013_CVPR_paper.html), [6] (https://www.osti.gov/biblio/1076723) and [7] (https://ieeexplore.ieee.org/abstract/document/6722493). These methods all showed varying levels of success with the work in [8] (https://arxiv.org/abs/1406.2952) achieving a classification success rate of 75%.

Data

The data set found here (https://www.kaggle.com/datasets/gpiosenka/100-bird-species) contained 95,376 images of 450 species, of which 70,626 were training images, 22,500 were validation images and 2,250 were testing images. Each image contained only one bird in each image and the bird typically took up at least 50% of the pixels in the image with each image already being scaled to 224 x 224 RGB pixels.

Modelling

The proposed classification model is a standard small convnet comprising multiple convolutional layers, multiple max-pooling layers, two dense layers and a dropout layer.

The basic architecture can be seen as [INPUT -> CONV2D -> MAXPOOL -> CONV2D -> MAXPOOL -> DENSE _> DROPOUT -> DENSE]. More specifically:

- INPUT: As we know all of the images are 224 x 244 pixels, with each pixel having red, green and blue channels, we can therefore state the input shape will be [224 x 224 x 3].
- CONV2D: This layer will compute the output of each of the neurons that are connected to local regions in the input. Each output neuron will be calculated by computing an elementwise dot product with the kernel and the region covered by the kernel. As we chose to use 64 filters, we know the output of this layer will be [224 x 224 x 64]. This layer uses the 'ReLU' activation function which computes max(0,x), essentially setting a hard minimum of 0. This also helps reduce computation time for the gradient as it is either 0 or 1 depending on the sign of the input.
- MAXPOOL: This layer downsamples the feature map passed into it. Using a pool size of 2 x 2, we effectively halve the spatial dimensions of the input in order to reduce the computational requirement. We can know the output layer to be [112 x 112 x 64]
- DENSE: These fully connected layers are used to compute the class scores with the final dense layer having the same number of neurons as there are classes in the data set. The second to last dense layer has a varying number of neurons, with the final value being found by utilizing the AWS HyperParameter Tuner.
- DROPOUT: This layer, added between the two final dense layers, attempts to reduce the overfitting of the model to the training set. These layers work by randomly ignoring the outputs of previous layers having the effect of allowing a single model architecture to simulate a large number of different architectures during its training.

In an attempt to reduce computational costs, early stopping was implemented. Early stopping stops training of the model when a monitored metric has stopped improving, which in this case was the model's "loss".

Batch training was also implemented to reduce the training time of the model. Graphics Processing Units (GPUs) are fantastic at computing nD-array operations and by leveraging this fact, we can instead feed in multiple images to the network at a time during training. Due to the matrix math computed, it doesn't really matter whether 100, 256, 2048 or even 10000 images are used per batch, as long as the GPU has enough memory to support such a large amount of data. The exact number of images is a hyperparameter, but for this testing, a batch size of 32 was used.

For the model testing, two optimizers were considered:

1. mini-batch gradient descent: this optimizer is regarded as the best variation of the gradient descent algorithms, being an improvement over both stochastic gradient descent and standard gradient descent. This method does have some challenges, with the largest being choosing the optimum learning rate. If the rate is too small then the algorithm will take a large amount of time to converge, and a learning rate too large may cause the algorithm to hop around the minima, never fully converging at it. This issue was mitigated by using the AWS HyperParamater Tuner to test a variety of learning rates.
2. Adam (ADAptive Moment estimation): this optimizer works by storing exponentially decaying past gradients as well as an exponentially decaying average of past squared gradients. This allows the optimizer to "slow down" as it reaches closer to the minima at a cost of being computationally expensive.

Local Model Development

Connect to s3 bucket

In [1]:
```
import sagemaker
# define environment variables
sess = sagemaker.Session()
role = sagemaker.get_execution_role()
bucket = sess.default_bucket()

print(bucket)
```

```
sagemaker-us-east-2-985637496371
```

Configure Imports

In [2]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib.pyplot import imread
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import time
import os
import itertools
import numpy as np
from sagemaker.tensorflow import TensorFlow
from tensorflow.keras.optimizers import Adam, Nadam, RMSprop, SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Dropout, Flatten, Input, Ma
xPooling2D

print("Tensorflow ver: " + tf.__version__)
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/cpu/lib/
python3.6/site-packages/tensorflow_core/__init__.py:1473: The name tf.estimat
or.inputs is deprecated. Please use tf.compat.v1.estimator.inputs instead.

Tensorflow ver: 1.15.5
```

Local constant declarations

In [3]:
```
training_local_path = './train/'
validation_local_path = './validation/'
test_local_path = './test/'
graph_path = os.makedirs("./output/graph/", exist_ok=True)

img_cols = 224
img_rows = 224
```

In [4]:
```
datagen = ImageDataGenerator(rescale=1.0/255.0)

train_batches = datagen.flow_from_directory(training_local_path , class_mode=
'categorical',
                                            target_size=(img_rows, img_cols))

val_batches = datagen.flow_from_directory(validation_local_path , class_mode=
'categorical',
                                          target_size=(img_rows, img_cols
))

test_batches = datagen.flow_from_directory(test_local_path , class_mode='categ
orical',
                                           target_size=(img_rows, img_cols
))
```

```
Found 1589 images belonging to 10 classes.
Found 50 images belonging to 10 classes.
Found 50 images belonging to 10 classes.
```

Visualise the classes

In [5]:

```
classes = list(test_batches.class_indices.keys())

import matplotlib.pyplot as plt
from matplotlib.pyplot import imread

fig, axs = plt.subplots(nrows = 5, ncols = 2, figsize=(10,10))
axs = axs.flatten()


for i in range(0,10):
    filename = test_local_path + classes[i] + '/'  + '1.jpg'
    img = imread(filename)
    axs[i].imshow(img)
    axs[i].axis("off")
    axs[i].set(title=classes[i] + " (" + str(i) + ")")
```

In [6]:
```
print('X_train shape:', train_batches[0][0].shape)
print('X_val shape:', val_batches[0][0].shape)
print('y_train shape:', train_batches[0][1].shape)
print('y_val shape:', val_batches[0][1].shape)
```

```
X_train shape: (32, 224, 224, 3)
X_val shape: (32, 224, 224, 3)
y_train shape: (32, 10)
y_val shape: (32, 10)
```

In [7]:
```
model = tf.keras.Sequential()
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', input_shape
=(img_rows, img_cols, 3)))
## downsample using a max pooling layer, which feeds into the next set of conv
olutional layers
model.add(MaxPooling2D(pool_size=(2,2)))
## 2nd convolultion layer
## convolutional layer with 32 filters
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
# flatten and classify
## flattern spacial information into a vector, and learn the final probability
distribution for each class
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation="softmax"))

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/cpu/lib/
python3.6/site-packages/tensorflow_core/python/ops/resource_variable_ops.py:1
630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resour
ce_variable_ops) with constraint is deprecated and will be removed in a futur
e version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

Mini-batch Descent Optimizer

In [8]:
```
model.compile(optimizer=SGD(lr=0.01, decay=0.1, momentum=0.1, nesterov=False),
loss='categorical_crossentropy', metrics=["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 32)      18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 93312)             0
_________________________________________________________________
dense (Dense)                (None, 32)                2986016
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330
=================================================================
Total params: 3,006,602
Trainable params: 3,006,602
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
model.fit(train_batches, validation_data=val_batches, epochs=2, steps_per_epoch=len(train_batches),
                          validation_steps=len(val_batches), verbose=1, callbacks=[ callback ])
```

```
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/__init__.py:163: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/horovod/tensorflow/__init__.py:189: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

[2022-10-30 05:28:20.975 ip-172-16-84-213:24351 INFO utils.py:27] RULE_JOB_STOP_SIGNAL_FILENAME: None
[2022-10-30 05:28:21.048 ip-172-16-84-213:24351 INFO profiler_config_parser.py:111] Unable to find config at /opt/ml/input/config/profilerconfig.json. Profiler is disabled.
WARNING:tensorflow:OMP_NUM_THREADS is no longer used by the default Keras config. To configure the number of threads, use tf.config.threading APIs.
Epoch 1/2
49/50 [============================>.] - ETA: 1s - loss: 2.2883 - acc: 0.1541
Epoch 1/2
50/50 [==============================] - 85s 2s/step - loss: 2.2887 - acc: 0.1536 - val_loss: 2.2742 - val_acc: 0.1600
Epoch 2/2
49/50 [============================>.] - ETA: 1s - loss: 2.2519 - acc: 0.1914
Epoch 1/2
50/50 [==============================] - 78s 2s/step - loss: 2.2524 - acc: 0.1907 - val_loss: 2.2509 - val_acc: 0.2000
```

Out[9]:
```
<tensorflow.python.keras.callbacks.History at 0x7f4bf43ddcf8>
```

In [10]:
```
train_scores = model.evaluate(train_batches, verbose=1)
validation_scores = model.evaluate(val_batches, verbose=1)

# print the final model metrics once the training job completes
print('=' * 25)
print('Mini-batch Descent')
print('=' * 25)
print('Training loss       :', train_scores[0])
print('Training accuracy   :', train_scores[1])
print('Validation loss       :', validation_scores[0])
print('Validation accuracy   :', validation_scores[1])
```

```
50/50 [==============================] - 29s 579ms/step - loss: 2.2373 - acc:
0.2901
2/2 [==============================] - 1s 460ms/step - loss: 2.2661 - acc: 0.
2000
=========================
Mini-batch Descent
=========================
Training loss       : 2.2372596740722654
Training accuracy   : 0.29011956
Validation loss       : 2.2661101818084717
Validation accuracy   : 0.2
```

Adam Optimizer

In [11]:
```
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 32)      18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 93312)             0
_________________________________________________________________
dense (Dense)                (None, 32)                2986016
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330
=================================================================
Total params: 3,006,602
Trainable params: 3,006,602
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
model.fit(train_batches, validation_data=val_batches, epochs=2, steps_per_epoch=len(train_batches),
                         validation_steps=len(val_batches), verbose=1, callbacks=[ callback ])
```

```
Epoch 1/2
49/50 [============================>.] - ETA: 1s - loss: 2.4397 - acc: 0.1240
Epoch 1/2
50/50 [==============================] - 82s 2s/step - loss: 2.4323 - acc: 0.1259 - val_loss: 2.1498 - val_acc: 0.2200
Epoch 2/2
49/50 [============================>.] - ETA: 1s - loss: 2.0402 - acc: 0.2415
Epoch 1/2
50/50 [==============================] - 84s 2s/step - loss: 2.0386 - acc: 0.2398 - val_loss: 2.0366 - val_acc: 0.4000
```

Out[12]:
```
<tensorflow.python.keras.callbacks.History at 0x7f4bf8d0eb70>
```

In [13]:
```
train_scores = model.evaluate(train_batches, verbose=1)
validation_scores = model.evaluate(val_batches, verbose=1)

# print the final model metrics once the training job completes
print('=' * 25)
print('ADAM')
print('=' * 25)
print('Training loss       :', train_scores[0])
print('Training accuracy   :', train_scores[1])
print('Validation loss       :', validation_scores[0])
print('Validation accuracy   :', validation_scores[1])
```

```
50/50 [==============================] - 28s 570ms/step - loss: 1.7857 - acc:
0.3952
2/2 [==============================] - 1s 474ms/step - loss: 2.0521 - acc: 0.
4000
=========================
ADAM
=========================
Training loss       : 1.7857248187065125
Training accuracy   : 0.39521712
Validation loss       : 2.0520676374435425
Validation accuracy   : 0.4
```

Adam outperforms MBD by almost 100% on validation accuracy. We shall use it when tuning the rest of the hyperparameters

Remote Deployment

Configure paths

In [14]:
```
prefix = "birds"
tensorflow_logs_path = "s3://{}/{}/logs".format(bucket, prefix)
training_input_path = "s3://{}/{}/train/".format(bucket, prefix)
validation_input_path = "s3://{}/{}/validation/".format(bucket, prefix)
testing_input_path = "s3://{}/{}/test/".format(bucket, prefix)
output_path = "s3://{}/{}/output".format(bucket, prefix)

print("SageMaker ver: " + sagemaker.__version__)

print("Bucket: {}".format(bucket))
print("Logs path: " + tensorflow_logs_path)
print("Training data location: " + training_input_path)
print("Validation data location: " + validation_input_path)
print("Testing data location: " + testing_input_path)
print("Output path: " + output_path)
print(role)
```

```
SageMaker ver: 2.113.0
Bucket: sagemaker-us-east-2-985637496371
Logs path: s3://sagemaker-us-east-2-985637496371/birds/logs
Training data location: s3://sagemaker-us-east-2-985637496371/birds/train/
Validation data location: s3://sagemaker-us-east-2-985637496371/birds/validat
ion/
Testing data location: s3://sagemaker-us-east-2-985637496371/birds/test/
Output path: s3://sagemaker-us-east-2-985637496371/birds/output
arn:aws:iam::985637496371:role/service-role/AmazonSageMaker-ExecutionRole-202
20908T141621
```

Hyperparamter Optimisation

In [15]:
```
from sagemaker.tuner import IntegerParameter, ContinuousParameter

# Define the range of hyperparameter values we will test
hyperparameter_ranges = {
    'epochs' :       IntegerParameter(5, 20),
    'learning-rate': ContinuousParameter(0.001, 0.1, scaling_type='ReverseLoga
rithmic'),
    'dense-layer':   IntegerParameter(10, 50)
}
```

In [16]:
```
# Metric used for evaluating training jobs
objective_metric_name = 'validation_accuracy'

# We wish to maximize the validation accuracy. If the above metric was loss, w
e would instead to minimize instead
objective_type = 'Maximize'
```

In [17]:
```
# define the metrics to be sent to CloudWatch
metric_definitions = [
    {'Name': 'training_loss',        'Regex': 'loss: ([0-9\\.]+)'},
    {'Name': 'training_accuracy',    'Regex': 'accuracy: ([0-9\\.]+)'},
    {'Name': 'validation_loss',      'Regex': 'val_loss: ([0-9\\.]+)'},
    {'Name': 'validation_accuracy',  'Regex': 'val_accuracy: ([0-9\\.]+)'},
    {'Name': 'training_precision',   'Regex': 'precision: ([0-9\\.]+)'}
]
```

In [18]:
```
# configure the estimator
tf_estimator = TensorFlow(
    entry_point = 'birds_CNN_tf2_adam.py',
    role = role,
    instance_count = 1,
    instance_type = 'ml.m4.xlarge',
    framework_version = '2.1.0',
    py_version = 'py3',
    script_mode = True,
    output_path = output_path,
    # Optional spot instance configuration
    use_spot_instances=True,        # Use spot instance
    max_run=3600,                   # Max training time
    max_wait=7200
)
```

In [21]:
```
from sagemaker.tuner import HyperparameterTuner

# we pass the tf_estimator to HyperparameterTuner(), and pass in the objective
_metric_name,
#objective_type and metric_definitions that we defined in the previous cell
tuner = HyperparameterTuner(tf_estimator,
                            objective_metric_name, #see cell above for definit
ion
                            hyperparameter_ranges, #see cell above for definit
ion
                            metric_definitions, #see cell above for definition
                            max_jobs=20,  #the maximum number of total trainin
g jobs to start - I set 4 just for illustration and time waiting
                            max_parallel_jobs=4, #Maximum number of parallel t
raining jobs to start
                            objective_type=objective_type,#see cell above for
 definition
                            base_tuning_job_name="Birds-CNN"
)
```

In [22]:
```
# fit the model in SageMaker
tuner.fit({
    'training': training_input_path,
    'validation': validation_input_path,
    'testing': testing_input_path
})
```

```
No finished training job found associated with this estimator. Please make su
re this estimator is only used for building workflow config

.....................................................................
```

In [25]:
```
import sagemaker
import boto3
from sagemaker.tuner import HyperparameterTuner
tuner = HyperparameterTuner.attach('Birds-CNN-221030-0553')
analytics = tuner.analytics()
df = analytics.dataframe()
df
```

Out[25]:

	dense-layer	epochs	learning-rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	Training
0	50.0	5.0	0.099901	Birds-CNN-221030-0553-020-d5220ee7	Completed	0.10	2 06:39
1	10.0	20.0	0.001251	Birds-CNN-221030-0553-019-a5311a1e	Completed	0.34	2 06:32
2	50.0	8.0	0.099527	Birds-CNN-221030-0553-018-319c3d6f	Completed	0.10	2 06:31
3	50.0	5.0	0.097363	Birds-CNN-221030-0553-017-1727e1cf	Completed	0.10	2 06:30
4	11.0	20.0	0.002084	Birds-CNN-221030-0553-016-cee2dbbc	Completed	0.10	2 06:26
5	50.0	5.0	0.018755	Birds-CNN-221030-0553-015-06f83a37	Completed	0.10	2 06:24
6	50.0	5.0	0.099576	Birds-CNN-221030-0553-014-dc2a9852	Completed	0.10	2 06:23
7	10.0	20.0	0.096505	Birds-CNN-221030-0553-013-c26b3b09	Completed	0.10	2 06:23
8	50.0	5.0	0.097807	Birds-CNN-221030-0553-012-d9bfc9f5	Completed	0.10	2 06:18
9	50.0	5.0	0.005934	Birds-CNN-221030-0553-011-43df5dd8	Completed	0.10	2 06:15
10	13.0	20.0	0.021335	Birds-CNN-221030-0553-010-2f306ea0	Completed	0.10	2 06:15
11	49.0	5.0	0.049363	Birds-CNN-221030-0553-009-68271a60	Completed	0.10	2 06:14
12	10.0	7.0	0.098433	Birds-CNN-221030-0553-008-040d064a	Completed	0.10	2 06:06
13	40.0	20.0	0.002831	Birds-CNN-221030-0553-007-eb43df90	Completed	0.10	2 06:06
14	14.0	18.0	0.076721	Birds-CNN-221030-0553-006-2c16e302	Completed	0.10	2 06:06
15	10.0	8.0	0.071262	Birds-CNN-221030-0553-005-5ba16cf4	Completed	0.10	2 06:05
16	42.0	13.0	0.057363	Birds-CNN-221030-0553-004-3e84d646	Completed	0.10	2 05:55

	dense-layer	epochs	learning-rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	Training
17	41.0	9.0	0.066555	Birds-CNN-221030-0553-003-364d2a57	Completed	0.10	2 05:56
18	31.0	6.0	0.004071	Birds-CNN-221030-0553-002-8b29a4ff	Completed	0.10	2 05:55
19	28.0	17.0	0.013250	Birds-CNN-221030-0553-001-8d636bd6	Completed	0.10	2 05:55

In [26]:
```
best_job = df.sort_values('FinalObjectiveValue', ascending=0)[:1] # pandas.df.sort_values
                                                                  # FinalObjectiveValue = Accuracy
best_job
```

Out[26]:

	dense-layer	epochs	learning-rate	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingS
1	10.0	20.0	0.001251	Birds-CNN-221030-0553-019-a5311a1e	Completed	0.34	20 06:32:5

In [27]:
```
# Pull out the name of the best job
best_job_name = best_job['TrainingJobName'].to_string(index=False).strip()
best_job_name
```

Out[27]: `'Birds-CNN-221030-0553-019-a5311a1e'`

Deploying the best model

In [32]:
```
endpoint_name = best_job_name + '-ep'

# Deploy the best model
best_model_predictor = tuner.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium',
    endpoint_name=endpoint_name)
```

```
2022-10-30 06:52:05 Starting - Preparing the instances for training
2022-10-30 06:52:05 Downloading - Downloading input data
2022-10-30 06:52:05 Training - Training image download completed. Training in
progress.
2022-10-30 06:52:05 Uploading - Uploading generated training model
2022-10-30 06:52:05 Completed - Training job completed

update_endpoint is a no-op in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

-----!
```

In [33]:
```
import sys
import random
import matplotlib.pyplot as plt

num_samples = 500
all_labels=[]
all_predicted_labels=[]


test_local_path = './test/'
test_batches = datagen.flow_from_directory(test_local_path , class_mode='categ
orical',
                                           target_size=(img_rows, img_cols),
batch_size=1)
```

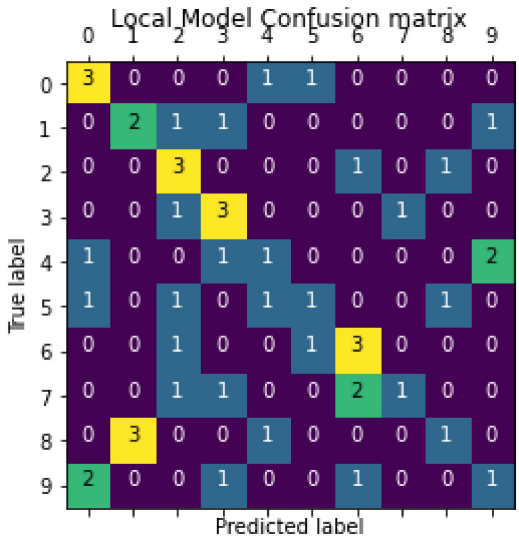
```
Found 50 images belonging to 10 classes.
```

In [34]:
```
num_samples = len(test_batches.filenames)
all_labels=[]
all_predicted_labels=[]

for i in range(0, num_samples):
    images = test_batches[i][0]
    labels = test_batches[i][1]
    prediction = best_model_predictor.predict(images)['predictions']
    prediction = np.array(prediction)
    predicted_labels = prediction.argmax(axis=1)
    all_labels.extend(labels.argmax(axis=1))
    all_predicted_labels.extend(predicted_labels)
```

In [35]:
```
import sklearn
import itertools
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(all_labels, all_predicted_labels)
plt.matshow(cm)
plt.title('Local Model Confusion matrix')
fmt = 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] < thresh else "black")
plt.ylabel('True label')
plt.xlabel('Predicted label')
classes = range(10) # Labels are sorted
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
plt.grid(False)
plt.show()
```



In [36]:
```
tuner.delete_endpoint()
```

```
The function delete_endpoint is a no-op in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

Model Evaluation and Deployment using AWS-SageMaker

The best hyper-parameter tuned model achieved a training accuracy of 76.626% and a validation accuracy of 34%. This is slightly worse than the local model developed in the SageMaker notebook. This best tuned model was deployed to an inference endpoint for testing with the test image set and achieved an accuracy of 38%. Unfortunately, this accuracy is quite low and deployment to the field will instead prove detrimental when trying to solve the problem at hand. A more complex model may be able to overcome the low accuracy, but further time will be needed to test this hypothesis.

Discuss the use of AWS SageMaker

The original instance type used for this notebook was an ml.t3.medium as the AWS free tier has 250 free hours. This instance has 2 vCPUs and 4 GiB of memory. This proved enough for training the local model, although when attempting to utilise ResNet152 during transfer learning, the instance would run out of memory. As such, the instance type was upgraded to an ml.t3.xlarge which has 4 vCPUs and 16GiB of memory. In total, the cost for all services (SageMaker, HyperParameter Tuning, Inference endpoints and EC2 training) came to $2.78 AUD and took ~45 mins to run with the largest time sink being the tuning. This cost is drastically smaller than if one were to upfront purchase the hardware required and proves the effectiveness of leveraging the cloud for one-off tasks that require large amounts of computation.

Transfer-learning and Model Comparison

The base model used for the transfer learning was ResNet152. This 152-layer deep residual network won 1st place on the ILSVRC 2015 classification task whilst being eight times deeper than VGG nets but still having an overall lower complexity. This base resnet was then appended with two fully connected layers. The first has 1000 neurons and the second has 10 as that is the number of classes we wish to predict. Once upgrading the instance type, training took very minimal time and with only 10 epochs was able to achieve a testing accuracy of 96%. Utilising the resnet based model, a bird species classifier which classifies the species of a bird based on an input image could be utilised in the field.

In [37]:
```
# configure the estimator
tf_estimator = TensorFlow(
    entry_point = 'resnet_CNN_tf2_transfer.py',
    role = role,
    instance_count = 1,
    instance_type = 'ml.c4.8xlarge',
    framework_version = '2.1.0',
    py_version = 'py3',
    script_mode = True,
    output_path = output_path,
    hyperparameters = { 'epochs': 10 },
    # Optional spot instance configuration
    use_spot_instances=True,        # Use spot instance
    max_run=7200,                   # Max training time
    max_wait=7200
)
```

In [38]:
```
# fit model
tf_estimator.fit({
    'training': training_input_path,
    'validation': validation_input_path,
    'testing': testing_input_path
})
```

```
2022-10-30 12:13:20 Starting - Starting the training job...
2022-10-30 12:13:45 Starting - Preparing the instances for trainingProfilerRe
port-1667132000: InProgress
.............
2022-10-30 12:15:50 Downloading - Downloading input data
2022-10-30 12:15:50 Training - Downloading the training image...
2022-10-30 12:16:04 Training - Training image download completed. Training in
progress.2022-10-30 12:16:08,294 sagemaker-containers INFO     Imported frame
work sagemaker_tensorflow_container.training
2022-10-30 12:53:13 Uploading - Uploading generated training model
2022-10-30 12:55:56 Completed - Training job completed
Training seconds: 2432
Billable seconds: 552
Managed Spot Training savings: 77.3%
```

In [39]:
```
import time
tf_endpoint_name = 'transfer-learning-endpoint-'+time.strftime("%Y-%m-%d-%H-%M
-%S", time.gmtime())

predictor = tf_estimator.deploy(initial_instance_count=1, # The initial number
of instances to run in the Endpoint created from this Model.
                                instance_type='ml.t2.2xlarge', # The EC2 in
stance type to deploy this Model to.
                                endpoint_name=tf_endpoint_name) # The name
 of the endpoint to create
```

```
update_endpoint is a no-op in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

-----!
```

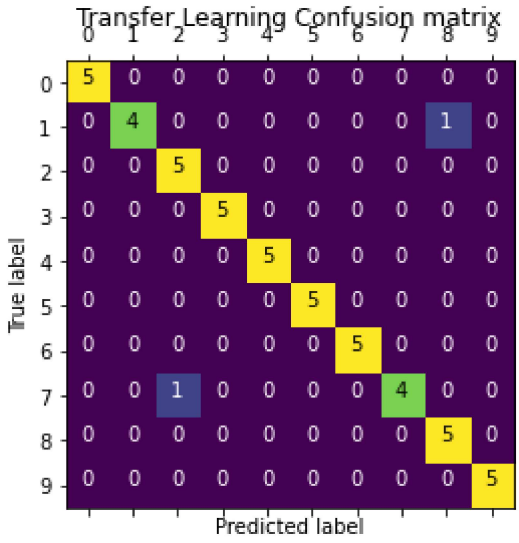
In [40]:
```
test_batches = datagen.flow_from_directory(test_local_path , class_mode='categ
orical',
                                           target_size=(img_rows, img_cols
), batch_size = 1)
num_samples = len(test_batches.filenames)
all_labels=[]
all_predicted_labels=[]

for i in range(0, num_samples):
    images = test_batches[i][0]
    labels = test_batches[i][1]
    prediction = predictor.predict(images)['predictions']
    prediction = np.array(prediction)
    predicted_labels = prediction.argmax(axis=1)
    all_labels.extend(labels.argmax(axis=1))
    all_predicted_labels.extend(predicted_labels)
```

```
Found 50 images belonging to 10 classes.
```

In [41]:
```
import sklearn
import itertools
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(all_labels, all_predicted_labels)
plt.matshow(cm)
plt.title('Transfer Learning Confusion matrix')
fmt = 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] < thresh else "black")
plt.ylabel('True label')
plt.xlabel('Predicted label')
classes = range(10) # Labels are sorted
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
plt.grid(False)
plt.show()
```



In [42]:
```
predictor.delete_endpoint()
```