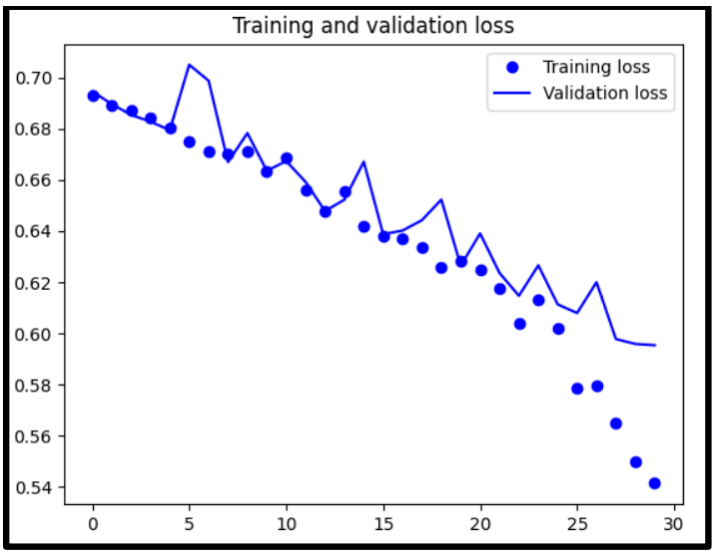
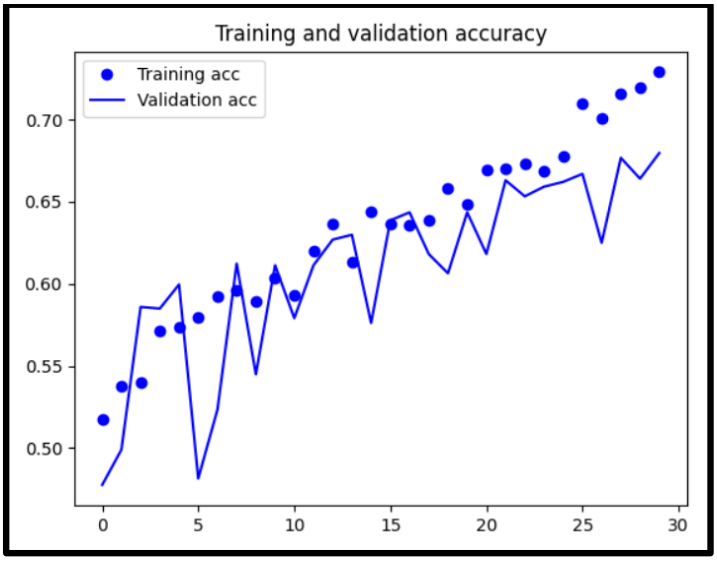
葉偉良 108065511

Report

1. cat_dog.py output.
   1.1 sample code's output
   a. loss



   b. accuracy

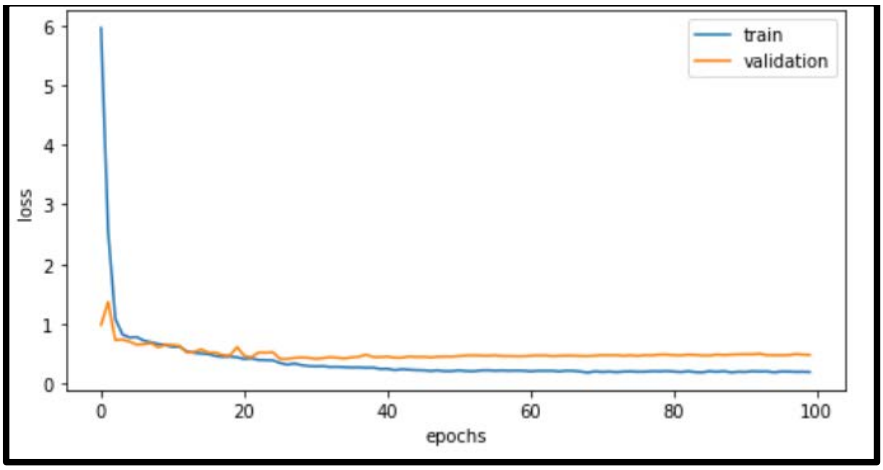


   c. validation accuracy

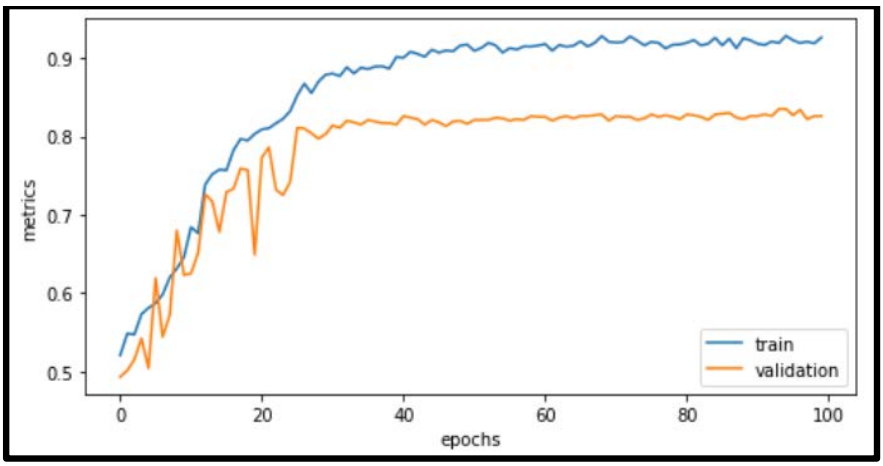
```
input shape: (150, 150)
output shape: 1
Found 1000 images belonging to 2 classes.
32/32 [==============================] - 3s 99ms/step - loss: 0.6144 - acc: 0.6930
test loss: 0.614391, test acc: 0.693000
```

1.2 my model's output

a. loss



b. accuracy



c. validation accuracy

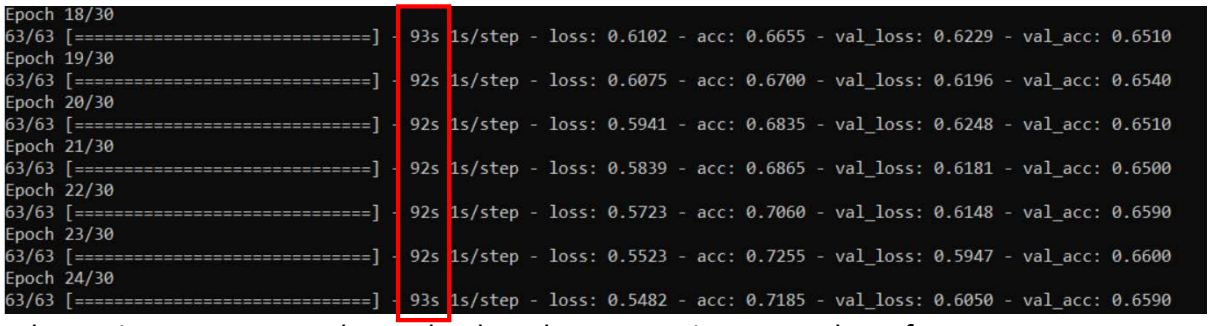
```
input shape: (180, 180)
output shape: 1
Found 1000 images belonging to 2 classes.
32/32 [==============================] - 20s 619ms/step - loss: 0.4490 -
binary_accuracy: 0.8510
test loss: 0.448996, test acc: 0.851000
```

2. use 2 VMs + horovod to run the code

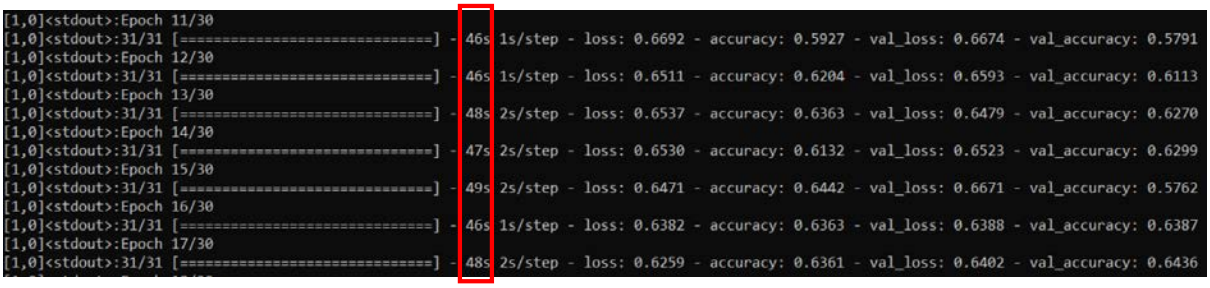
proof:

```
[1,0]<stdout>:Epoch 9/30
[1,0]<stdout>:31/31 [==============================] - 46s 1s/step - loss: 0.6721 - accuracy: 0.5897 - val_loss: 0.6783 - val_accuracy: 0.5449
[1,0]<stdout>:Epoch 10/30
[1,0]<stdout>:31/31 [==============================] - 47s 2s/step - loss: 0.6645 - accuracy: 0.6035 - val_loss: 0.6636 - val_accuracy: 0.6113
[1,0]<stdout>:Epoch 11/30
[1,0]<stdout>:31/31 [==============================] - 46s 1s/step - loss: 0.6692 - accuracy: 0.5927 - val_loss: 0.6674 - val_accuracy: 0.5791
[1,0]<stdout>:Epoch 12/30
[1,0]<stdout>:31/31 [==============================] - 46s 1s/step - loss: 0.6511 - accuracy: 0.6204 - val_loss: 0.6593 - val_accuracy: 0.6113
[1,0]<stdout>:Epoch 13/30
[1,0]<stdout>:31/31 [==============================] - 48s 2s/step - loss: 0.6537 - accuracy: 0.6363 - val_loss: 0.6479 - val_accuracy: 0.6270
[1,0]<stdout>:Epoch 14/30
[1,0]<stdout>:31/31 [==============================] - 47s 2s/step - loss: 0.6530 - accuracy: 0.6132 - val_loss: 0.6523 - val_accuracy: 0.6299
[1,0]<stdout>:Epoch 15/30
[1,0]<stdout>:31/31 [==============================] - 49s 2s/step - loss: 0.6471 - accuracy: 0.6442 - val_loss: 0.6671 - val_accuracy: 0.5762
[1,0]<stdout>:Epoch 16/30
[1,0]<stdout>:31/31 [==============================] - 46s 1s/step - loss: 0.6382 - accuracy: 0.6363 - val_loss: 0.6388 - val_accuracy: 0.6387
[1,0]<stdout>:Epoch 17/30
```

3. Speedup gained when using 2 VMs
   a. When using only one VM, each epoch takes about 93 seconds



   b. When using two VMs, each epoch takes almost two times speed up of one VM, which is around 47 seconds

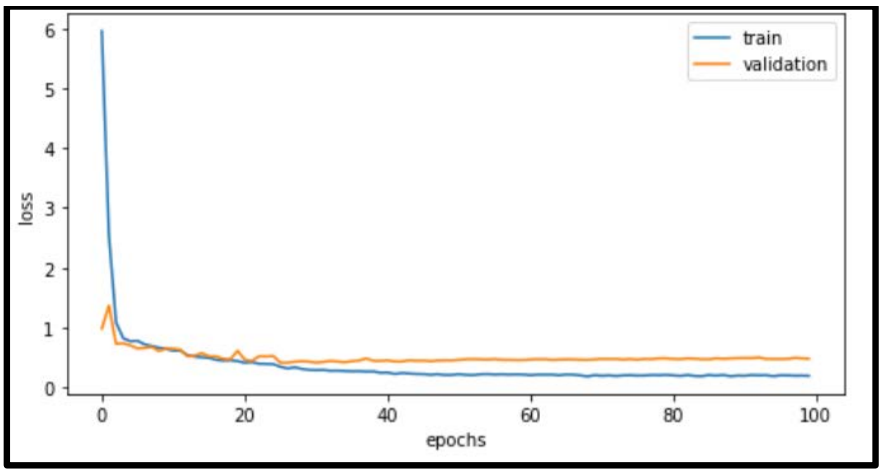


4. Why can't we achieve 100% scaling efficiency? How to improve?
   The reason of why it couldn't achieve 100% scaling efficiency would be there still needed to take a bit of time on the communication between those two VMs such as sharing parameters, weights. To improve the efficiency, we could use the distributed algorithm that reduce the time used on communication between VMs.

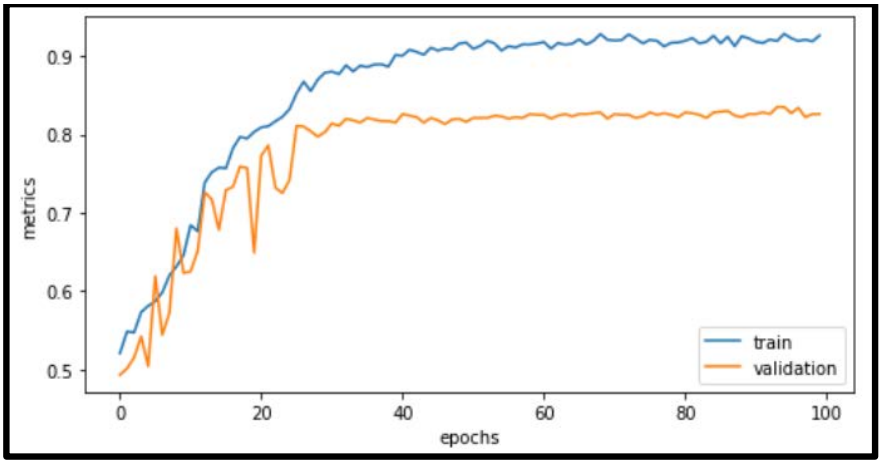
5. My model.summary()

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 180, 180, 64)      4864
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 90, 90, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 90, 90, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 90, 64)        102464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 45, 45, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        102464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 23, 64)        256
_________________________________________________________________
flatten (Flatten)            (None, 33856)             0
_________________________________________________________________
dense (Dense)                (None, 384)               13001088
_________________________________________________________________
dropout (Dropout)            (None, 384)               0
_________________________________________________________________
dense_1 (Dense)              (None, 192)               73920
_________________________________________________________________
dense_2 (Dense)              (None, 64)                12352
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 13,300,033
Trainable params: 13,299,649
Non-trainable params: 384
_________________________________________________________________
```

6. My training curve of accuracy and loss
   a. loss



b. accuracy



7. Modifications I did to make the model better.
   a. Model architecture

   As shown in (5.), I modified the model architecture with:

   - An ***input layers*** with accepting the input shape of (180, 180, 3)
   - 3 times of ***convolutional layers***, ***max-pooling layers***, and ***batch normalization*** with the same settings.
   - ***Fatten layers*** to make the output of convolutional layers can be connected with full connected layers (Dense)
   - 4 of the ***fully connected layers*** (Dense) with one dropout layers
   - And the ***output layers*** with activation function softmax.

   CODE:

```
def build_model(name):
    inputs = keras.Input(shape=(IMG_WIDTH, IMG_HEIGHT, 3))
    x = layers.Conv2D(64, (5, 5), padding='same', activation='relu')(inputs)
    x = layers.MaxPool2D(pool_size=3,strides=2,padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv2D(64, (5, 5), padding='same', activation='relu')(x)
    x = layers.MaxPool2D(pool_size=3,strides=2,padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.Conv2D(64, (5, 5), padding='same', activation='relu')(x)
    x = layers.MaxPool2D(pool_size=3,strides=2,padding='same')(x)
    x = layers.BatchNormalization()(x)

    x = layers.Flatten()(x)
    x = layers.Dense(384, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(192, activation='relu')(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dense(32, activation='relu')(x)
    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs, outputs, name=name)
    model.summary()

    return model
```

b. Optimizer

There are two optimizers I have tried, ***Adam*** and ***RMSprop***(**Root-Mean-Square**). The result came out with the ***RMSprop*** would make the model achieve higher accuracy without overfitting issue. Thus I choose ***RMSprop*** as my optimizer.

CODE:

```
from tensorflow.keras.optimizers import RMSprop
```

```
optimizer = RMSprop(lr=0.001, rho=0.95, epsilon=1e-08, decay=0.0)
```

c. Learning Rate

Instead of giving a fixed number of the learning rate, I use a technique called ***learning rate reduction***. This technique can modify the value of the learning rate based on the situation faced on each training epoch.

CODE:

```
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_binary_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

d. Data Augmentation

Doing rotation, zoom, shear, horizontal flip, weight shift, height shift on the training datasets to make model learning stronger information from the dataset.

CODE:

```
#generate training data with augmentation
train_datagen = ImageDataGenerator( rotation_range=15,
                                    rescale=1./255,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True,
                                    #vertical_flip=True,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
                                    #channel_shift_range=0.1,
                                    #brightness_range=[0.1, 10]
                                  )
```

e. Adding a dropout layer in model architecture

I have faced the overfitting issue first, hence I added a dropout layers with parameter 0.5 to make model not rely on the training dataset too much.

f. The value of epochs and batch size, the size of the training images

I have changed the value of epoch to 100, the batch size becomes 64, and the size of the training images with height 180 and width 180.