Deep Learning

Summative assessment

Coursework 2

Instructions

This coursework is released on **Wednesday 21st February 9.00** and is due by **Wednesday 6th March 23.59**. It is worth **40%** of your overall mark. There are 4 questions in this assessment, and a total of 100 marks are available. **You should attempt to answer all questions.** In addition to the total number of marks per question below, an additional 10 marks is available for presentation and clarity/quality of code.

This assessment assesses your ability to design, implement, train and evaluate a deep learning model for a classification task using multimodal data.

You can make imports as and when you need them throughout the notebook, and add code cells where necessary. Make sure your notebook executes correctly in sequence before submitting.

Submission instructions

The submission for this assessment will consist of a notebook (.ipynb file) and a PDF submission.

Ensure your notebook executes correctly in order. Save your notebook .ipynb file **after you have executed it** (so that outputs are all showing). It is recommended to also submit a PDF copy of your executed notebook, in case the .ipynb file is corrupted for some reason.

Upload a zip file containing your notebook and separate PDF file(s) to Coursera by the deadline above.

In [ ]:
```
# You will need the following imports for this assessment. You can make addition

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

from tensorflow.keras.layers import (Layer, Input, Dense, GRU, Embedding, Conv2D
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import CSVLogger, ModelCheckpoint
from tabulate import tabulate
```

In [ ]:
```
# You will need access to a GPU for this coursework

print(tf.config.list_physical_devices('GPU'))
tf.keras.backend.clear_session()
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

The CLEVR Dataset

This assessment makes use of the CLEVR Dataset. This dataset is a visual question answering dataset, and consists of images with corresponding text questions and answers about the image.

- Johnson, J., Hariharan, B., van der Maaten, L., Li, F.-F., Zitnick, C. L. & Girshick, R. (2016), "CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1988-1997.

The original dataset consists of a training set of 70,000 images and 699,989 questions, a validation set of 15,000 images and 149,991 questions, and a test set of 15,000 images and 14,988 questions. In this coursework you will work with a subset of the training and validation splits, which have been preprocessed and prepared for you.

The data is stored in TFRecord format, which is a data format that is efficient for TensorFlow to work with. You can read about the TFRecord format here if you are interested, but there is no need to find out about TFRecord for this assessment. The code to read in the TFRecord data to Dataset objects is provided for you below.

In [ ]:
```
train_ds = tf.data.TFRecordDataset([os.path.join('../../../data_CW2/data', 'trai
                                    for f in os.listdir(os.path.join('../../../d
val_ds = tf.data.TFRecordDataset([os.path.join('../../../data_CW2/data', 'val',
                                    for f in os.listdir(os.path.join('../../../d
```

In [ ]:
```
# The following helper function will parse the TFRecord files to return a dictio

def parse_function(example_proto):
    features = {
        "image": tf.io.FixedLenFeature((), tf.string),
        "question": tf.io.VarLenFeature(dtype=tf.string),
        "answer": tf.io.VarLenFeature(dtype=tf.string)
    }
    parsed_features = tf.io.parse_single_example(serialized=example_proto, featu
    parsed_features["question"] = tf.sparse.to_dense(parsed_features["question"]
    parsed_features["answer"] = tf.sparse.to_dense(parsed_features["answer"])
    image = tf.io.decode_raw(parsed_features["image"], tf.int32)
    image = tf.reshape(image, [224, 224, 3])
    parsed_features["image"] = image
    return parsed_features
```

In [ ]:
```
train_ds = train_ds.map(parse_function)
val_ds = val_ds.map(parse_function)
```

In [ ]:
```
train_ds.element_spec
```

Out[ ]:
```
{'answer': TensorSpec(shape=(None,), dtype=tf.string, name=None),
 'question': TensorSpec(shape=(None,), dtype=tf.string, name=None),
 'image': TensorSpec(shape=(224, 224, 3), dtype=tf.int32, name=None)}
```

Your task in this assessment is to develop a deep learning model to predict the answer for a given question about an image.

You will need to implement special customised layers and a sophisticated model architecture, making use of both CNN and RNN models. You will process the data, train and evaluate the specified model, and then write a proposal for your own modified architecture.

Question 1 (Total 15 marks)

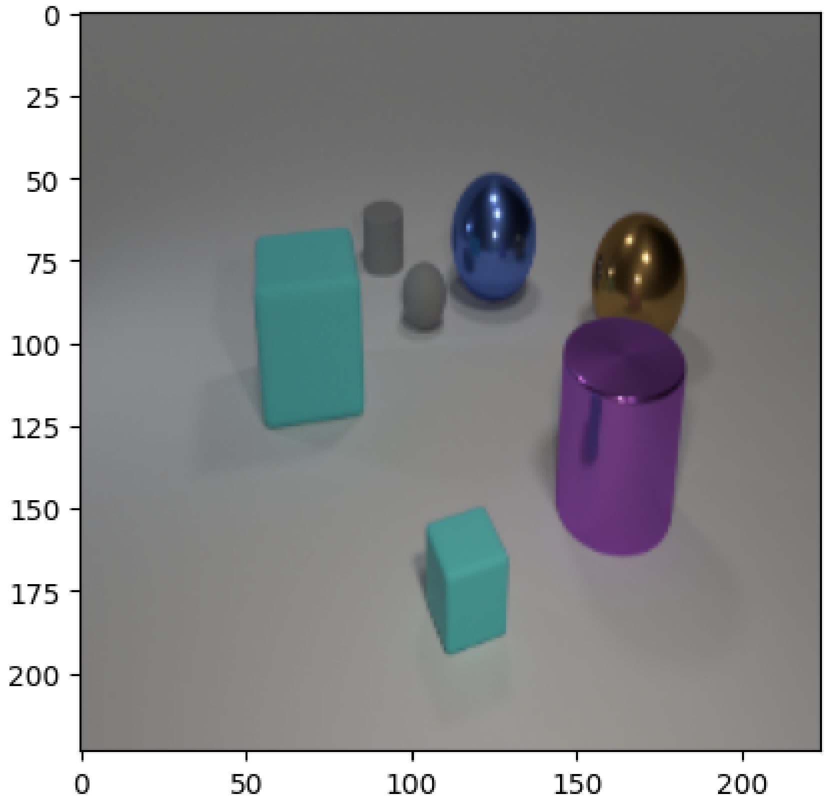
The training and validation datasets both return dictionaries with keys `"image"`, `"question"` and `"answer"`. For each image, there are multiple questions and answers. The question and answer entries in the dictionary are both lists of strings of the same length, with aligned questions and answers for the given image. The image entry is a 224x224x3 integer Tensor. These images have been resized from the original size of 480x320, so they appear slightly stretched (this can be ignored).

a) Inspect the contents of the dataset by displaying at least one image and it's corresponding questions and answers.

(3 marks)

In [ ]:
```
for ele in train_ds.take(1):
    eg_q, eg_a, eg_i = ele["question"], ele["answer"], ele["image"]
```

In [ ]:
```
# First, we show the example image
plt.imshow(eg_i)
plt.show()
```



In [ ]:
```
#Next, we show the example questions and answers for the image
l = eg_q.shape[0]
for combo in range(l):
    tf.print("Question:", eg_q[combo])
    tf.print("Answer:", eg_a[combo], "\n")
```

```
Question: "The blue metal object is what size?"
Answer: "large"

Question: "What is the material of the small cyan cube?"
Answer: "large"

Question: "What is the material of the small cyan cube?"
Answer: "rubber"

Question: "Is the size of the cyan thing that is in front of the purple cylinder
the same as the small matte ball?"
Answer: "yes"

Question: "What number of things are either gray balls or big yellow metal object
s?"
Answer: "1"

Question: "There is a rubber object that is the same color as the large block; wh
at is its shape?"
Answer: "cube"

Question: "There is a shiny thing that is on the left side of the big brown thing
and behind the large cyan thing; what size is it?"
Answer: "large"

Question: "What number of tiny yellow metal cylinders are there?"
Answer: "0"

Question: "How many blocks are either tiny cyan rubber things or small matte thin
gs?"
Answer: "1"

Question: "There is a blue ball right of the cyan block that is behind the purple
thing; how many brown objects are to the right of it?"
Answer: "1"

Question: "The cylinder that is the same size as the blue metal ball is what colo
r?"
Answer: "purple"
```

b) The training and validation Datasets should be processed as follows:

- The image pixel values should be scaled to the interval $[0, 1]$.
- The answers should be (sparse) encoded as integer labels. You will need to compute the total number of distinct answers to do this.
- The questions should be tokenized and represented as a sequence of integer tokens. The questions should be split on whitespace and standardized by lowercasing and removing punctuation.
- A single question-answer pair should be uniformly sampled from the available questions and answers for each image (so each image should appear exactly once per epoch with a single question-answer pair).
- The inputs to the model will be the question and the image. The targets will be the answer. Process the Datasets so that they return a tuple of 2 elements corresponding to inputs and targets.
- Shuffle the training Dataset, and batch both Datasets with batch size 64.

Print out the number of distinct answer labels, as well as the number of tokens in the vocabulary computed from the questions.

Print out the element_spec of one of the Datasets after processing.

(12 marks)

In [ ]:
```
#Scale the images
def image_scale(dic):
    im = dic["image"]
    im = im / 255
    dic["image"] = im
    return dic

train_ds = train_ds.map(image_scale)
val_ds = val_ds.map(image_scale)
```

In [ ]:
```
### Sparse encode answers ###
#find number of unique answers
text_vectorization_ans = TextVectorization(standardize='lower_and_strip_punctuat

#get answers for train set
answers = train_ds.map(lambda l: l["answer"])

#adapt on training set only
text_vectorization_ans.adapt(answers)

vocab_ans_size = text_vectorization_ans.vocabulary_size()
vocab_ans = text_vectorization_ans.get_vocabulary()
print(f"There are {vocab_ans_size} distinct answers including unknown and a plac
print(vocab_ans)

#this shows that there are 30 unique answers, including unknown and placeholder
```

```
There are 30 distinct answers including unknown and a placeholder
['', '[UNK]', 'no', 'yes', '1', '0', 'rubber', 'metal', 'small', 'large', '2', 'c
ylinder', 'sphere', 'cube', '3', 'blue', 'yellow', 'brown', 'gray', 'purple', 'cy
an', 'red', 'green', '4', '5', '6', '7', '8', '9', '10']
```

In [ ]:
```
#do sparse encoding as requested, we have [UNK] for OOV indices
stringlookup = StringLookup(vocabulary = vocab_ans, num_oov_indices=0)

def convert_labels(element):
    #update the dictionary given
    element["answer"] = stringlookup(element["answer"])
    return element

train_ds = train_ds.map(convert_labels)

#this will be the same for validation

val_ds = val_ds.map(convert_labels)
```

In [ ]:
```
### Now to tokenize all of the questions ###
text_vectorization_q = TextVectorization(standardize='lower_and_strip_punctuatio
#adapt to the questions of the training set
questions = train_ds.map(lambda l: l["question"])
text_vectorization_q.adapt(questions)
```

```python
vocab_q_size = text_vectorization_q.vocabulary_size()
vocab_q = text_vectorization_q.get_vocabulary()
```

In [ ]:
```python
def tokenize_q(element):
    q = element["question"]
    element["question"] = text_vectorization_q(q)
    return element
#map it
train_ds = train_ds.map(tokenize_q)
val_ds = val_ds.map(tokenize_q)
```

In [ ]:
```python
### Now to sample randomly for each image ###

def sample_image(element):
    q, a = element["question"], element["answer"]
    #10 questions and answers per one
    num_qs = tf.shape(q)[0]
    #have to use tensorflow operation or it picks the same for all
    idx = tf.random.uniform(shape=(), minval=0, maxval=num_qs, dtype=tf.int32)
    q = q[idx,:]
    a = a[idx]
    element["question"] = q
    element["answer"] = a
    return element

train_ds = train_ds.map(sample_image)
val_ds = val_ds.map(sample_image)
```

In [ ]:
```python
### Now do inputs and targets ###
def inputs_and_targets(element):
    inputs = (element["question"], element["image"])
    targets = element["answer"]
    return inputs, targets

train_ds = train_ds.map(inputs_and_targets)
val_ds = val_ds.map(inputs_and_targets)
```

In [ ]:
```python
### Now shuffle and batch ###
#only shuffe training set as stated on ED
batch_size = 64
train_ds = train_ds.shuffle(100)

#padded batching to make sure the inputs are all of the same shape for each batc
train_ds = train_ds.padded_batch(batch_size)
val_ds = val_ds.padded_batch(batch_size)
```

In [ ]:
```python
#finally, prefetch
train_ds = train_ds.prefetch(tf.data.AUTOTUNE)
val_ds = val_ds.prefetch(tf.data.AUTOTUNE)
```

In [ ]:
```python
### Print off element specs ###

print(train_ds.element_spec)
```

```
((TensorSpec(shape=(None, None), dtype=tf.int64, name=None), TensorSpec(shape=(No
ne, 224, 224, 3), dtype=tf.float64, name=None)), TensorSpec(shape=(None,), dtype=
tf.int64, name=None))
```

Question 2 (Total 35 marks)

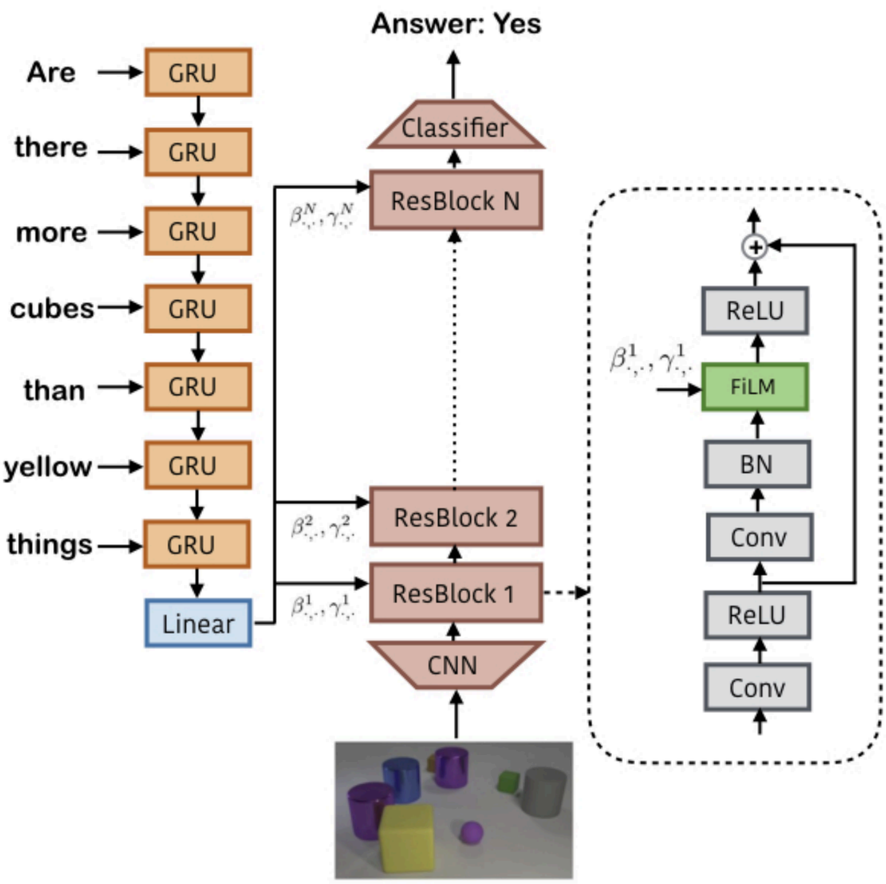
The model that you will implement for the visual question answering task was first proposed in the paper

- Perez, E., Strub, F., de Vries, H. & Courville, A. (2018), "FiLM: visual reasoning with a general conditioning layer", in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA.

The main idea is the introduction of a specialised layer called a FiLM layer (Feature-wise Linear Modulation). The purpose of this layer is to modify the predictions that are made by a CNN prediction model (the central stack coloured in brown in the figure below). The CNN prediction model takes the image as input, and outputs a categorical distribution over the set of possible answers.

The FiLM layer uses information stored in a vector embedding (which comes from the question text) to modify the post-activations of the CNN prediction model. This vector embedding is produced by a gated recurrent unit (GRU) network (referred to in the original paper as the FiLM generator) as the final hidden layer representation after processing the input question. This vector embedding is also referred to as the conditioning signal.

The overall model architecture is shown in the figure below:



Overall model architecture

The question is tokenized, and learned embeddings are processed sequentially by the GRU network/FiLM generator. There are potentially multiple FiLM layers within the CNN prediction model. Each FiLM layer uses the GRU embedding $\mathbf{q}$ (the conditioning signal) to modify the output of a convolutional layer within the CNN prediction model, as described in part c).

a) Implement the FiLM generator as a 2-layer stacked GRU network, using an embedding dimension of 64, and 128 neurons for both of the layers of the GRU. The network should output the final 128-dimensional embedding. Print the model summary.

(3 marks)

In [ ]:
```
#we implement a function to get the generator for use later
def get_gru():
    gru_network = Sequential([
        #embed
        Embedding(input_dim = vocab_q_size,output_dim = 64,mask_zero = True),
        GRU(units = 128, return_sequences = True),
        GRU(units = 128)
    ], name = "gru_network")
    return gru_network
```

In [ ]:
```
gru_network = get_gru()
gru_network.summary()
```

```
Model: "gru_network"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 64)          5248

 gru (GRU)                   (None, None, 128)         74496

 gru_1 (GRU)                 (None, 128)               99072

=================================================================
Total params: 178,816
Trainable params: 178,816
Non-trainable params: 0
_________________________________________________________________
```


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 64)          5248

 gru (GRU)                   (None, None, 128)         74496

 gru_1 (GRU)                 (None, 128)               99072

=================================================================
Total params: 178,816
Trainable params: 178,816
Non-trainable params: 0
_________________________________________________________________
```


b) The first block of the CNN prediction model is a feature extractor CNN which does not make use of the conditioning signal $\mathbf{q}$ from the GRU network. This block takes the image as input, and passes it through two sub-blocks, each consisting of the following layers:

- A 2D convolutional layer with 128 filters, a 4x4 kernel, 2x2 strides, 'SAME' padding, and no activation function
- A batch normalisation layer
- An element-wise ReLU activation

Implement the feature extractor CNN and print the model summary.

(2 marks)

In [ ]:
```
def get_cnn_feature_extractor():
    cnn_feature_extractor = Sequential([
        #sub block one
        Conv2D(filters = 128, kernel_size = (4,4), strides = (2,2), padding = "s
        BatchNormalization(),
        Activation("relu"),
        #sub block two
        Conv2D(filters = 128, kernel_size = (4,4), strides = (2,2), padding = "s
        BatchNormalization(),
        Activation("relu")], name = "feature_extractor")
    return cnn_feature_extractor
```

In [ ]:
```
cnn_feature_extractor = get_cnn_feature_extractor()
print(cnn_feature_extractor.summary())
```

```
Model: "feature_extractor"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 112, 112, 128)     6272      
                                                                 
 batch_normalization (BatchN  (None, 112, 112, 128)    512       
 ormalization)                                                   
                                                                 
 activation (Activation)     (None, 112, 112, 128)     0         
                                                                 
 conv2d_1 (Conv2D)           (None, 56, 56, 128)       262272    
                                                                 
 batch_normalization_1 (Batc  (None, 56, 56, 128)      512       
 hNormalization)                                                 
                                                                 
 activation_1 (Activation)   (None, 56, 56, 128)       0         
                                                                 
=================================================================
Total params: 269,568
Trainable params: 269,056
Non-trainable params: 512
_________________________________________________________________
None
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 112, 112, 128)     6272      
                                                                 
 batch_normalization (BatchN  (None, 112, 112, 128)    512       
 ormalization)                                                   
                                                                 
 activation (Activation)     (None, 112, 112, 128)     0         
                                                                 
 conv2d_1 (Conv2D)           (None, 56, 56, 128)       262272    
                                                                 
 batch_normalization_1 (Batc  (None, 56, 56, 128)      512       
 hNormalization)                                                 
                                                                 
 activation_1 (Activation)   (None, 56, 56, 128)       0         
                                                                 
=================================================================
Total params: 269,568
Trainable params: 269,056
Non-trainable params: 512
_________________________________________________________________
None
```

c) Implement a custom layer class for the FiLM layer as described below. This class should subclass the base `Layer` class in the `tensorflow.keras.layers` module.

This layer will need to take two inputs when it is called: the conditioning signal $\mathbf{q}$, as well as the previous convolutional layer output $\mathbf{h}$.

The FiLM layer passes the conditioning signal $\mathbf{q}$ output by the GRU FiLM generator through a linear layer (dense layer with no activation function) to produce γ and β:

$$\gamma = \mathrm{Linear}(\mathbf{q}) \qquad \beta = \mathrm{Linear}(\mathbf{q}).$$

Both γ and β are vectors, with length equal to the number of feature maps (or channels) in the output of a convolutional layer $\mathbf{h}$. These post-activations are then modulated via the feature-wise affine transformation:

$$\mathrm{FiLM}(\mathbf{h}|\gamma,\beta)_{h,w,c} = \gamma_c \mathbf{h}_{h,w,c} + \beta_c,$$

where the subscripts h, w, c index the height, width and channel dimensions respectively.

Create an instance of your custom layer class and test it on some dummy inputs to verify it works as expected.

(10 marks)

In [ ]:
```
#the output channels of the convolutional network is 128
output_channels = 128
class FiLM(Layer):
    def __init__(self):
        super().__init__()
        #initialise the linear transformations so the weights can then be learne
        #we only want the given term, not the bias too
        self.dense_gamma = Dense(output_channels, name = "gamma", use_bias = Fal
        self.dense_beta = Dense(output_channels, name = "beta", use_bias = False


    def call(self,q,h):
        gamma = self.dense_gamma(q)
        beta = self.dense_beta(q)
        #return broadcasted result, could have done [None, etc]
        return tf.expand_dims(tf.expand_dims(gamma,1),1) * h + tf.expand_dims(tf
```

In [ ]:
```
film = FiLM()
```

In [ ]:
```
#take dummy data
#take the first example batch for each
for ele in train_ds.take(1):
    inputs, outputs = ele

print(film(gru_network(inputs[0]),cnn_feature_extractor(inputs[1])))
film(gru_network(inputs[0]), cnn_feature_extractor(inputs[1])).shape
#this gives us the shape that we expect
```

```
tf.Tensor(
[[[[-0.00394472 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00356205]
   [-0.00387035 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00333774]
   [-0.00387053 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00333535]
   ...
   [-0.00386849 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00333785]
   [-0.00386934 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00333516]
   [-0.00382091 -0.0047873   0.00574359 ...  0.00930811  0.00647941
     0.00326752]]

  [[-0.00405313 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00333494]
   [-0.00408463 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00408581 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   ...
   [-0.00407638 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.0040761  -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00397438 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]]

  [[-0.00405175 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00332924]
   [-0.00408567 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00408816 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   ...
   [-0.00407645 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00407405 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00397277 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]]

  ...

  [[-0.00407795 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00333979]
   [-0.00411702 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00411754 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   ...
   [-0.00420045 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00420023 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]
   [-0.00404527 -0.0047873   0.00574359 ...  0.00930811  0.00644133
     0.00326752]]

  [[-0.00407952 -0.0047873   0.00574359 ...  0.00930811  0.00644133
```

```
    0.0033407 ]
  [-0.00411677 -0.0047873   0.00574359 ...  0.00930811  0.00644133
    0.00326752]
  [-0.00411936 -0.0047873   0.00574359 ...  0.00930811  0.00644133
    0.00326752]
  ...
  [-0.00420274 -0.0047873   0.00574359 ...  0.00930811  0.00644133
    0.00326752]
  [-0.0042018  -0.0047873   0.00574359 ...  0.00930811  0.00644133
    0.00326752]
  [-0.00404611 -0.0047873   0.00574359 ...  0.00930811  0.00644133
    0.00326752]]

 [[-0.00406273 -0.0047133   0.00574359 ...  0.00930811  0.00644133
    0.00330697]
  [-0.00418538 -0.00467665  0.00574359 ...  0.00930811  0.00644133
    0.00351153]
  [-0.00418763 -0.00467571  0.00574359 ...  0.00930811  0.00644133
    0.00351371]
  ...
  [-0.00428937 -0.0046438   0.00574359 ...  0.00930811  0.00644133
    0.00357979]
  [-0.00428789 -0.00464222  0.00574359 ...  0.00930811  0.00644133
    0.00357961]
  [-0.0041776  -0.0047873   0.00574359 ...  0.00930811  0.00644133
    0.00372033]]]


[[[-0.00583219  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00305887]
  [-0.00577592  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00296013]
  [-0.0057761   0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00295896]
  ...
  [-0.00577449  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.0029599 ]
  [-0.00577465  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00295817]
  [-0.0057386   0.00295467  0.01056644 ...  0.00278401  0.01070744
    0.00292881]]

 [[-0.00591444  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00295807]
  [-0.00593778  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.0059386   0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  ...
  [-0.00593225  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00593294  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00585456  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]]

 [[-0.00591312  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00295774]
  [-0.00593891  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
```

```
  [-0.00594043  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  ...
  [-0.00593279  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00593111  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00585437  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]]

 ...

 [[-0.00593534  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00296098]
  [-0.00596495  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00596539  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  ...
  [-0.00600676  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00600438  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00589585  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]]

 [[-0.00593646  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00296146]
  [-0.00596451  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00596636  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  ...
  [-0.00600684  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00600657  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]
  [-0.00589659  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00292881]]

 [[-0.00592325  0.00324172  0.01056644 ...  0.00278401  0.01064468
    0.00294627]
  [-0.00601576  0.00338123  0.01056644 ...  0.00278401  0.01064468
    0.00303727]
  [-0.00601892  0.00338346  0.01056644 ...  0.00278401  0.01064468
    0.00303824]
  ...
  [-0.00606986  0.00346893  0.01056644 ...  0.00278401  0.01064468
    0.00305746]
  [-0.00606789  0.003462    0.01056644 ...  0.00278401  0.01064468
    0.00305821]
  [-0.00598964  0.00295467  0.01056644 ...  0.00278401  0.01064468
    0.00311585]]]


[[[ 0.00287333 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
   -0.00015911]
  [ 0.0029704  -0.00525974 -0.00897319 ... -0.00062636  0.00395878
   -0.00022363]
  [ 0.00296969 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
```

```
  -0.00022439]
 ...
 [ 0.0029727  -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00022394]
 [ 0.00297286 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00022342]
 [ 0.00303582 -0.00525974 -0.00897319 ... -0.00062636  0.00405972
  -0.00024376]]

[[ 0.00272628 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00022489]
 [ 0.00268728 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00268317 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 ...
 [ 0.0026917  -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00269212 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00283008 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]]

[[ 0.00272745 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00022504]
 [ 0.00268391 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00268359 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 ...
 [ 0.00269228 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00269227 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00283015 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]]

...

[[ 0.00268714 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00022202]
 [ 0.00263402 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00263338 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 ...
 [ 0.00251741 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00252131 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00273404 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]]

[[ 0.0026867  -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00022214]
 [ 0.00263402 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 [ 0.00263079 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
  -0.00024376]
 ...
```

```
  [ 0.00251216 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
   -0.00024376]
  [ 0.00251501 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
   -0.00024376]
  [ 0.0027299  -0.00525974 -0.00897319 ... -0.00062636  0.00395878
   -0.00024376]]

 [[ 0.00270851 -0.0051221  -0.00897319 ... -0.00062636  0.00395878
   -0.00023217]
  [ 0.00254233 -0.00505625 -0.00897319 ... -0.00062636  0.00395878
   -0.00017207]
  [ 0.00253696 -0.00505243 -0.00897319 ... -0.00062636  0.00395878
   -0.00017108]
  ...
  [ 0.00238369 -0.00498743 -0.00897319 ... -0.00062636  0.00395878
   -0.00014909]
  [ 0.00238864 -0.0049892  -0.00897319 ... -0.00062636  0.00395878
   -0.00014977]
  [ 0.00254472 -0.00525974 -0.00897319 ... -0.00062636  0.00395878
   -0.00010739]]]


...


[[[ 0.00580762 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00223777]
  [ 0.00589055 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00273239]
  [ 0.00588953 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00274173]
  ...
  [ 0.00589359 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00273543]
  [ 0.00589271 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00273765]
  [ 0.00594547 -0.01166438 -0.003584   ...  0.0083888   0.00280475
   -0.00289159]]

 [[ 0.00568606 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.0027421 ]
  [ 0.00565106 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00564888 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  ...
  [ 0.00565994 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00566071 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00577359 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]]

 [[ 0.00568711 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00275253]
  [ 0.00564977 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.0056472  -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  ...
```

```
  [ 0.00565958 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00566109 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00577517 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]]

 ...

 [[ 0.0056502  -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00272334]
  [ 0.00560749 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00560463 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  ...
  [ 0.00555989 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00556179 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.0057166  -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]]

 [[ 0.00564866 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00272542]
  [ 0.0056069  -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00560401 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  ...
  [ 0.00555863 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.0055594  -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]
  [ 0.00571614 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00289159]]

 [[ 0.00566772 -0.01137029 -0.003584   ...  0.0083888   0.00270107
   -0.00279867]
  [ 0.00552808 -0.01122838 -0.003584   ...  0.0083888   0.00270107
   -0.00232713]
  [ 0.00552447 -0.01122408 -0.003584   ...  0.0083888   0.00270107
   -0.00232068]
  ...
  [ 0.00546736 -0.01116488 -0.003584   ...  0.0083888   0.00270107
   -0.00224949]
  [ 0.00547068 -0.01116667 -0.003584   ...  0.0083888   0.00270107
   -0.00225152]
  [ 0.00558223 -0.01166438 -0.003584   ...  0.0083888   0.00270107
   -0.00195994]]]


[[[ 0.00082704 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
   -0.00214028]
  [ 0.00090541 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
   -0.00245634]
  [ 0.0009055  -0.00763328 -0.00537682 ...  0.00956351  0.00616101
   -0.00246074]
  ...
  [ 0.00090936 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
```

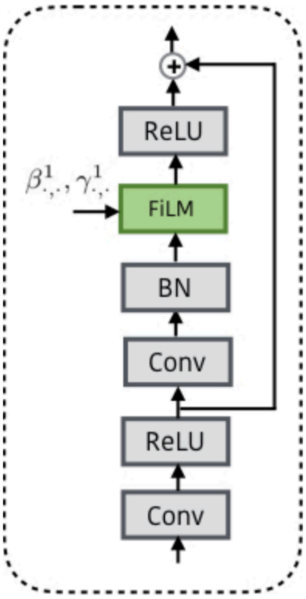
```
  -0.00245663]
 [ 0.00090765 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00246039]
 [ 0.00095843 -0.00763328 -0.00537682 ...  0.00956351  0.00621143
  -0.00255625]]

[[ 0.00071205 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00246346]
 [ 0.00067909 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00067662 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 ...
 [ 0.00068721 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00068702 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00079514 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]]

[[ 0.00071437 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.0024632 ]
 [ 0.00067788 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00067676 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 ...
 [ 0.00068863 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.0006885  -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00079696 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]]

...

[[ 0.00068331 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00245338]
 [ 0.00064284 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00064294 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 ...
 [ 0.00059286 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00059538 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00074295 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]]

[[ 0.0006828  -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00245076]
 [ 0.0006424  -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00064144 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 ...
 [ 0.00059071 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]
 [ 0.00059225 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
```

```
  -0.00255625]
 [ 0.00074163 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00255625]]

[[ 0.0007012  -0.00733482 -0.00537682 ...  0.00956351  0.00616101
  -0.00249776]
 [ 0.00057038 -0.00718473 -0.00537682 ...  0.00956351  0.00616101
  -0.00220887]
 [ 0.00056819 -0.00718678 -0.00537682 ...  0.00956351  0.00616101
  -0.00220477]
 ...
 [ 0.0005036  -0.0071065  -0.00537682 ...  0.00956351  0.00616101
  -0.00214887]
 [ 0.00050719 -0.00711393 -0.00537682 ...  0.00956351  0.00616101
  -0.0021498 ]
 [ 0.00061324 -0.00763328 -0.00537682 ...  0.00956351  0.00616101
  -0.00196714]]]


[[[ 0.00602927 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.0025749 ]
  [ 0.00607975 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00238059]
  [ 0.00607959 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00237709]
  ...
  [ 0.00608116 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00237933]
  [ 0.00608077 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00237781]
  [ 0.00611371 -0.00569264 -0.00626942 ...  0.01119442  0.0068908
    0.00231804]]

 [[ 0.00595444 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00237682]
  [ 0.00593342 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00593286 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  ...
  [ 0.00593926 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00593958 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00600895 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]]

 [[ 0.00595648 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00237375]
  [ 0.00593292 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00593104 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  ...
  [ 0.00593918 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00594014 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00601005 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]]
```

```
...

 [[ 0.00593421 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00238363]
  [ 0.00590706 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00590667 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  ...
  [ 0.00586314 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00586494 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00596749 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]]

 [[ 0.00593397 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00238361]
  [ 0.00590716 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00590549 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  ...
  [ 0.00586143 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00586339 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]
  [ 0.00596658 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.00231804]]

 [[ 0.00594553 -0.00572014 -0.00626942 ...  0.01119442  0.00679171
    0.00235237]
  [ 0.00585966 -0.00573388 -0.00626942 ...  0.01119442  0.00679171
    0.00253638]
  [ 0.00585728 -0.005734   -0.00626942 ...  0.01119442  0.00679171
    0.00253867]
  ...
  [ 0.00580148 -0.00574336 -0.00626942 ...  0.01119442  0.00679171
    0.0025903 ]
  [ 0.00580437 -0.00574293 -0.00626942 ...  0.01119442  0.00679171
    0.00258686]
  [ 0.00587851 -0.00569264 -0.00626942 ...  0.01119442  0.00679171
    0.0027033 ]]]], shape=(64, 56, 56, 128), dtype=float32)
```

Out[ ]:
```
TensorShape([64, 56, 56, 128])
```

d) The second main block of the CNN network consists of a number of ResBlocks. Each ResBlock consists of the following layers:



- A 1x1 convolutional layer with 128 channels and ReLU activation function
- A 3x3 convolutional layer with 128 channels and no activation function
- A BatchNormalization layer, where the usual γ and β parameters are not used
- a FiLM layer, that also uses the conditioning signal $\mathbf{q}$ from the GRU network
- An elementwise ReLU activation function
- The output is then added to the output of the first convolutional layer

Each convolutional layer uses 'SAME' padding.

Implement the ResBlock as another custom layer. Similar to the FiLM layer, this layer will also need to take two inputs when it is called: the conditioning signal $\mathbf{q}$, as well as the previous convolutional layer output $\mathbf{h}$.

Create an instance of your custom layer class and test it on some dummy inputs to verify it works as expected.

(8 marks)

In [ ]:
```
class ResBlock(Layer):
    def __init__(self):
        super().__init__()
        #dont need to give conv the input dimension
        self.conv1 = Conv2D(128, (1,1), activation = "relu", padding = "same")
        self.conv2 = Conv2D(128,(3,3), padding = "same")
        #turn the bn parameters off for the bn layer preceeding the film layer
        self.bn = BatchNormalization(center = False, scale = False)
        self.film = FiLM()
        self.relu = Activation("relu")
    def call(self, q, h):
        #send the inputs forward, noting that the first convolutional layer is a
        h = self.conv1(h)
        #save the output of the first convolutional layer for later
        conv1_h = h
        #proceed forward
        h = self.conv2(h)
        h = self.bn(h)
        h = self.film(q,h)
        h = self.relu(h)
```

```
        #now combine
        h = h + conv1_h
        return h
```

In [ ]:
```
#We will pass in the first example input again as before:
resblock = ResBlock()
print(resblock(gru_network(inputs[0]),cnn_feature_extractor(inputs[1])))
resblock(gru_network(inputs[0]),cnn_feature_extractor(inputs[1])).shape
#again we have the output dimension that we expect
```

```
tf.Tensor(
[[[[0.00238402 0.00740612 0.01181587 ... 0.02839302 0.01096677
    0.03112508]
   [0.00238679 0.00737248 0.03101223 ... 0.04915722 0.00929459
    0.01126014]
   [0.00238517 0.00735669 0.03069619 ... 0.04861517 0.00957751
    0.01114128]
   ...
   [0.00238486 0.00735818 0.02971518 ... 0.04711586 0.00852732
    0.01057689]
   [0.00237354 0.00735996 0.02945353 ... 0.04700648 0.00875173
    0.01056059]
   [0.00237347 0.00736589 0.03252681 ... 0.0239896  0.02310978
    0.01273261]]

  [[0.00235578 0.02687984 0.         ... 0.04914786 0.0452711
    0.        ]
   [0.00234912 0.00735517 0.01721121 ... 0.07691336 0.06987898
    0.        ]
   [0.00235232 0.00731229 0.01732242 ... 0.07720447 0.07021696
    0.        ]
   ...
   [0.00235331 0.00731583 0.01630605 ... 0.07388415 0.06744223
    0.        ]
   [0.00234766 0.00733835 0.01642369 ... 0.07393333 0.06750298
    0.        ]
   [0.00236107 0.00732006 0.01511    ... 0.04664368 0.06300981
    0.01124781]]

  [[0.00235672 0.02665679 0.         ... 0.04895755 0.04542186
    0.        ]
   [0.0023505  0.00737148 0.0169715  ... 0.07681071 0.06986933
    0.        ]
   [0.00235241 0.00732737 0.0172965  ... 0.07736591 0.06972582
    0.        ]
   ...
   [0.00235366 0.00733215 0.01658427 ... 0.07348751 0.06729924
    0.        ]
   [0.00234819 0.00736743 0.01642547 ... 0.07249316 0.06676513
    0.        ]
   [0.00235588 0.00734854 0.01489207 ... 0.04588902 0.06221021
    0.01103464]]

  ...

  [[0.00235424 0.02938802 0.         ... 0.05475099 0.05047004
    0.        ]
   [0.00234742 0.00736803 0.01892425 ... 0.0858385  0.07772639
    0.        ]
   [0.00234951 0.00731734 0.01918677 ... 0.08644113 0.0781361
    0.        ]
   ...
   [0.002342   0.00729243 0.02416422 ... 0.11149176 0.09984761
    0.        ]
   [0.00233395 0.00734877 0.02386223 ... 0.11105616 0.09957284
    0.        ]
   [0.00234519 0.00732005 0.02136073 ... 0.06844611 0.09230162
    0.01569422]]

  [[0.00236288 0.02980271 0.         ... 0.05495037 0.05062986
```

```
  0.        ]
 [0.00235265 0.00731592 0.01886692 ... 0.08580619 0.07782081
  0.        ]
 [0.00235449 0.00727084 0.01914601 ... 0.08642755 0.0780226
  0.        ]
 ...
 [0.00234811 0.00723234 0.02383451 ... 0.11198259 0.10003593
  0.        ]
 [0.00234249 0.0073032  0.02393341 ... 0.11147305 0.0999337
  0.        ]
 [0.00234133 0.00728038 0.02120058 ... 0.06820326 0.09273011
  0.01588407]]

[[0.0023527  0.05472969 0.         ... 0.07684372 0.06030758
  0.        ]
 [0.00234577 0.02817956 0.         ... 0.09863919 0.05382103
  0.        ]
 [0.00234388 0.02841766 0.         ... 0.09893718 0.05401202
  0.        ]
 ...
 [0.00233447 0.033781   0.         ... 0.12751661 0.06965279
  0.        ]
 [0.00234155 0.03377992 0.         ... 0.12699123 0.06932327
  0.        ]
 [0.00234212 0.00728942 0.00316852 ... 0.08715645 0.04980999
  0.02530522]]]


[[[0.         0.00220489 0.01171892 ... 0.02735783 0.01365202
   0.03112002]
  [0.         0.00229524 0.03125626 ... 0.04829182 0.01168403
   0.01146616]
  [0.         0.00233859 0.03065521 ... 0.04745027 0.01231307
   0.01126932]
  ...
  [0.         0.00233731 0.02968283 ... 0.04638257 0.0113985
   0.01064011]
  [0.         0.00233218 0.02962018 ... 0.04637388 0.01130462
   0.01064806]
  [0.         0.00231595 0.03262522 ... 0.02311293 0.02573417
   0.0128212 ]]

 [[0.         0.02167704 0.         ... 0.0485298  0.0476457
   0.        ]
  [0.         0.00234276 0.01698292 ... 0.07586648 0.07244816
   0.        ]
  [0.         0.00245875 0.01719686 ... 0.07606488 0.0725311
   0.        ]
  ...
  [0.         0.00245288 0.01645061 ... 0.07380445 0.07049514
   0.        ]
  [0.         0.00239075 0.0163292  ... 0.07363252 0.07052425
   0.        ]
  [0.         0.00244056 0.0150001  ... 0.04615968 0.06600688
   0.0112715 ]]

 [[0.         0.02152782 0.         ... 0.0484199  0.04780148
   0.        ]
  [0.         0.00230087 0.01702475 ... 0.07625681 0.07222912
   0.        ]
```

```
  [0.         0.00241862 0.01745887 ... 0.0765591  0.07248618
   0.        ]
  ...
  [0.         0.00240944 0.01660084 ... 0.07347338 0.07017403
   0.        ]
  [0.         0.0023107  0.01643308 ... 0.07244889 0.07037652
   0.        ]
  [0.         0.00236235 0.01499489 ... 0.04523328 0.06535212
   0.01119632]]

 ...

 [[0.         0.02477142 0.         ... 0.05473053 0.05350676
   0.        ]
  [0.         0.00231207 0.0192732  ... 0.08638686 0.08105816
   0.        ]
  [0.         0.00245012 0.01931106 ... 0.08705556 0.08155241
   0.        ]
  ...
  [0.         0.00249405 0.02276917 ... 0.10314593 0.09600396
   0.        ]
  [0.         0.00235296 0.02234777 ... 0.10249674 0.0955563
   0.        ]
  [0.         0.00242189 0.02011802 ... 0.06291582 0.08854125
   0.01493077]]

 [[0.         0.02482053 0.         ... 0.05482187 0.05347268
   0.        ]
  [0.         0.00245173 0.01941348 ... 0.08634123 0.08049785
   0.        ]
  [0.         0.00257632 0.0192501  ... 0.08662827 0.08137578
   0.        ]
  ...
  [0.         0.00264773 0.02260121 ... 0.10379731 0.09631018
   0.        ]
  [0.         0.00246718 0.02261628 ... 0.1032041  0.09586715
   0.        ]
  [0.         0.00252389 0.01979966 ... 0.0627837  0.08912482
   0.01488704]]

 [[0.         0.04997498 0.         ... 0.07688484 0.06365696
   0.        ]
  [0.         0.02362497 0.         ... 0.09894791 0.05655579
   0.        ]
  [0.         0.02375112 0.         ... 0.09949754 0.05739389
   0.        ]
  ...
  [0.         0.02743156 0.         ... 0.11864568 0.06764531
   0.        ]
  [0.         0.02684649 0.         ... 0.11781493 0.06741903
   0.        ]
  [0.         0.00249993 0.00298481 ... 0.08037337 0.04908285
   0.02370244]]]


[[[0.         0.00299992 0.01547005 ... 0.03299639 0.01119312
   0.03661362]
  [0.         0.00304057 0.03444455 ... 0.05364388 0.00959735
   0.01657631]
  [0.         0.00305983 0.03404577 ... 0.0533434  0.00946885
```

```
  0.01646038]
 ...
 [0.         0.00305953 0.03342824 ... 0.0523627  0.00907549
  0.01621904]
 [0.         0.00305821 0.03366975 ... 0.05218126 0.00875142
  0.0162283 ]
 [0.         0.00305072 0.03655641 ... 0.02928723 0.02342014
  0.01851437]]

[[0.         0.02252472 0.00363791 ... 0.05358768 0.04551133
  0.00544661]
 [0.         0.00306178 0.02065921 ... 0.08105348 0.06983365
  0.00561276]
 [0.         0.0031143  0.02037257 ... 0.08121607 0.06990489
  0.00564595]
 ...
 [0.         0.00311325 0.02012458 ... 0.07957923 0.06855194
  0.00564638]
 [0.         0.00308381 0.02020655 ... 0.07948469 0.0683076
  0.0056381 ]
 [0.         0.0031071  0.01878732 ... 0.05211066 0.06365614
  0.01681262]]

[[0.         0.02215731 0.00360225 ... 0.05366911 0.04544051
  0.00545444]
 [0.         0.00304208 0.02035506 ... 0.08100917 0.06967926
  0.00563447]
 [0.         0.0030957  0.02060485 ... 0.08122952 0.06953852
  0.0056491 ]
 ...
 [0.         0.00309259 0.02024597 ... 0.07923645 0.06814267
  0.00564675]
 [0.         0.00304803 0.01993266 ... 0.07867928 0.0683237
  0.00560869]
 [0.         0.00307228 0.01885022 ... 0.05131777 0.06317091
  0.01684437]]

...

[[0.         0.02528911 0.00361797 ... 0.06036478 0.05142774
  0.00544683]
 [0.         0.00304792 0.02302633 ... 0.09229089 0.07936684
  0.00565155]
 [0.         0.00311182 0.02306065 ... 0.09271221 0.079603
  0.00567012]
 ...
 [0.         0.00314437 0.02848171 ... 0.1194134  0.10322348
  0.00571659]
 [0.         0.00307366 0.02824103 ... 0.11878749 0.1023962
  0.00565804]
 [0.         0.00310835 0.02572502 ... 0.07456689 0.09462798
  0.02204124]]

[[0.         0.02555062 0.00351959 ... 0.0604711  0.05160175
  0.00544089]
 [0.         0.00311332 0.02295063 ... 0.09242307 0.07904736
  0.00556094]
 [0.         0.00316997 0.02268372 ... 0.09269015 0.07960729
  0.00557483]
 ...
```

```
  [0.         0.00322382 0.02848181 ... 0.12111047 0.10427719
   0.00559378]
  [0.         0.00313354 0.02825471 ... 0.11970197 0.10356817
   0.00552451]
  [0.         0.00316112 0.02582786 ... 0.0750791  0.09570441
   0.0219717 ]]

 [[0.         0.0512614  0.00343254 ... 0.08275502 0.06165801
   0.00574494]
  [0.         0.02455574 0.00341904 ... 0.10518574 0.05476795
   0.00583054]
  [0.         0.02462618 0.00337342 ... 0.10605904 0.05554134
   0.00582036]
  ...
  [0.         0.03073356 0.00334209 ... 0.13828139 0.07298538
   0.00591603]
  [0.         0.03046336 0.0033061  ... 0.13734068 0.07210744
   0.00580258]
  [0.         0.0031501  0.00704092 ... 0.0951681  0.05154075
   0.03223541]]]


...


[[[0.         0.00885601 0.01197588 ... 0.03072223 0.01461334
   0.03103027]
  [0.         0.00896122 0.03130733 ... 0.05165828 0.01271469
   0.01119749]
  [0.         0.00901091 0.03099994 ... 0.05116972 0.01305767
   0.01128634]
  ...
  [0.         0.00900747 0.02986004 ... 0.04967009 0.01214216
   0.0106719 ]
  [0.         0.00900272 0.02984051 ... 0.04962475 0.01221223
   0.01041936]
  [0.         0.00898351 0.03314376 ... 0.02691752 0.02651292
   0.01281651]]

 [[0.         0.02828696 0.0003956  ... 0.05165301 0.04879986
   0.        ]
  [0.         0.0090159  0.01752012 ... 0.07916089 0.07337773
   0.        ]
  [0.         0.00915095 0.01770938 ... 0.07956159 0.07371049
   0.        ]
  ...
  [0.         0.00914186 0.01693847 ... 0.07638625 0.07122387
   0.        ]
  [0.         0.00906876 0.01675251 ... 0.07653113 0.07092561
   0.        ]
  [0.         0.00912821 0.01557243 ... 0.04931051 0.06662904
   0.01127556]]

 [[0.         0.02807327 0.00038641 ... 0.05138602 0.04879059
   0.        ]
  [0.         0.00896509 0.01727972 ... 0.07943708 0.07318965
   0.        ]
  [0.         0.00910518 0.01769817 ... 0.08007047 0.07331505
   0.        ]
  ...
```

```
  [0.         0.00909151 0.01675466 ... 0.07613038 0.07072397
   0.        ]
  [0.         0.0089786  0.01697329 ... 0.07577472 0.07036491
   0.        ]
  [0.         0.0090368  0.01532133 ... 0.04846954 0.06598749
   0.01102041]]

 ...

 [[0.         0.03137003 0.00039049 ... 0.05864755 0.05532047
   0.        ]
  [0.         0.00898191 0.02002123 ... 0.09066223 0.08334447
   0.        ]
  [0.         0.00914352 0.02039733 ... 0.09100103 0.08357841
   0.        ]
  ...
  [0.         0.00918032 0.02241049 ... 0.1034662  0.0939673
   0.        ]
  [0.         0.00902152 0.02227821 ... 0.10305425 0.0941404
   0.        ]
  [0.         0.00909958 0.01996978 ... 0.06469236 0.08731812
   0.01440071]]

 [[0.         0.03136168 0.0003653  ... 0.05869425 0.05524378
   0.        ]
  [0.         0.00914849 0.01980033 ... 0.09048905 0.08309542
   0.        ]
  [0.         0.00929217 0.01995357 ... 0.09101393 0.08374403
   0.        ]
  ...
  [0.         0.0093526  0.0223468  ... 0.10387729 0.09461708
   0.        ]
  [0.         0.00915122 0.02232195 ... 0.10335312 0.09421802
   0.        ]
  [0.         0.00921424 0.01982036 ... 0.06464534 0.08783173
   0.01451179]]

 [[0.         0.05741983 0.00034259 ... 0.08129182 0.06557021
   0.        ]
  [0.         0.03048915 0.00033927 ... 0.10339957 0.05831003
   0.        ]
  [0.         0.03068107 0.00032677 ... 0.10416193 0.05901013
   0.        ]
  ...
  [0.         0.03351098 0.00032375 ... 0.11884881 0.06677789
   0.        ]
  [0.         0.0329426  0.0003149  ... 0.11800324 0.06635072
   0.        ]
  [0.         0.00918975 0.00349984 ... 0.08186638 0.04860743
   0.02324478]]]


[[[0.         0.00423521 0.01435095 ... 0.03324113 0.01167772
   0.03081993]
  [0.         0.00429899 0.03304302 ... 0.05388795 0.01033861
   0.01130634]
  [0.         0.00432979 0.03294295 ... 0.05339023 0.01036061
   0.01109659]
  ...
  [0.         0.00432645 0.03215983 ... 0.05167244 0.00991354
```

```
  0.01064237]
 [0.         0.00432396 0.03196356 ... 0.05173619 0.00975109
  0.01065577]
 [0.         0.00431263 0.0350318  ... 0.02899347 0.02398953
  0.01273878]]

[[0.         0.02364218 0.00249495 ... 0.05375263 0.04595984
  0.        ]
 [0.         0.00433203 0.01968155 ... 0.08138738 0.07052757
  0.        ]
 [0.         0.00441518 0.01970173 ... 0.08147069 0.07094143
  0.        ]
 ...
 [0.         0.00440913 0.01870269 ... 0.07859266 0.06824314
  0.        ]
 [0.         0.00436425 0.01893977 ... 0.07857507 0.0679467
  0.        ]
 [0.         0.0044003  0.01733555 ... 0.05156443 0.06365261
  0.01103574]]

[[0.         0.02356135 0.00247752 ... 0.05349686 0.0460667
  0.        ]
 [0.         0.00430035 0.01954221 ... 0.08130591 0.07056147
  0.        ]
 [0.         0.00438518 0.01970455 ... 0.08146767 0.07039187
  0.        ]
 ...
 [0.         0.00437735 0.01910332 ... 0.07873073 0.0676797
  0.        ]
 [0.         0.004308   0.01895684 ... 0.0776021  0.06791726
  0.        ]
 [0.         0.00434555 0.01744294 ... 0.05053022 0.06315413
  0.01111026]]

...

[[0.         0.02592293 0.00248479 ... 0.05985795 0.05141244
  0.        ]
 [0.         0.00430874 0.02176193 ... 0.0910499  0.07891358
  0.        ]
 [0.         0.00440693 0.02158869 ... 0.09159113 0.07907697
  0.        ]
 ...
 [0.         0.00443248 0.02426722 ... 0.10571749 0.09131081
  0.        ]
 [0.         0.00433666 0.02411431 ... 0.1052313  0.09119292
  0.        ]
 [0.         0.00438456 0.02210938 ... 0.06696714 0.08483231
  0.01433174]]

[[0.         0.02630848 0.00243701 ... 0.06004551 0.051617
  0.        ]
 [0.         0.00440868 0.02165043 ... 0.090951   0.07884769
  0.        ]
 [0.         0.00449498 0.02140731 ... 0.09135413 0.07917023
  0.        ]
 ...
 [0.         0.00453799 0.02434973 ... 0.10622079 0.09195315
  0.        ]
 [0.         0.00441489 0.02435572 ... 0.1057864  0.09146832
```

```
   0.        ]
  [0.         0.00445496 0.02212986 ... 0.06701748 0.08507313
   0.01456409]]

 [[0.         0.05165825 0.00239518 ... 0.08180213 0.0615091
   0.        ]
  [0.         0.02517821 0.00238941 ... 0.10359139 0.05440095
   0.        ]
  [0.         0.02539335 0.0023663  ... 0.10410058 0.05494338
   0.        ]
  ...
  [0.         0.02865877 0.00235953 ... 0.12110415 0.06408783
   0.        ]
  [0.         0.02829929 0.00234254 ... 0.12042265 0.06385475
   0.        ]
  [0.         0.00443908 0.0055862  ... 0.08419426 0.04617721
   0.02310874]]]


[[[0.00085535 0.         0.0165361  ... 0.03788054 0.01861388
   0.03121988]
  [0.0008605  0.         0.0354887  ... 0.0584588  0.01703603
   0.01140761]
  [0.0008574  0.         0.03507965 ... 0.05812498 0.01740423
   0.01143108]
  ...
  [0.0008571  0.         0.03437009 ... 0.05666432 0.01627635
   0.01046875]
  [0.00083694 0.         0.03420723 ... 0.05649384 0.0164152
   0.01048422]
  [0.00083682 0.         0.03698212 ... 0.03347114 0.03093731
   0.01280143]]

 [[0.00080531 0.01935463 0.00471624 ... 0.05862226 0.05319984
   0.        ]
  [0.00079348 0.         0.02179868 ... 0.08601663 0.07737125
   0.        ]
  [0.00079921 0.         0.02187367 ... 0.08659489 0.07770739
   0.        ]
  ...
  [0.00080085 0.         0.02096915 ... 0.08351351 0.07493938
   0.        ]
  [0.00079082 0.         0.0211213  ... 0.08348894 0.075138
   0.        ]
  [0.00081472 0.         0.01964401 ... 0.05613969 0.07070442
   0.01120846]]

 [[0.00080699 0.01917287 0.00465413 ... 0.05852341 0.05304213
   0.        ]
  [0.00079605 0.         0.02167629 ... 0.08628818 0.07768514
   0.        ]
  [0.00079931 0.         0.02175438 ... 0.08691069 0.07744486
   0.        ]
  ...
  [0.00080164 0.         0.02115845 ... 0.08326468 0.07483429
   0.        ]
  [0.00079173 0.         0.02075073 ... 0.08225093 0.07482181
   0.        ]
  [0.00080535 0.         0.01960933 ... 0.05541626 0.07008049
   0.01106608]]
```

```
...

 [[0.00080182 0.02248261 0.0046817  ... 0.06556614 0.0591934
   0.        ]
  [0.00078911 0.         0.02408627 ... 0.09707483 0.08706815
   0.        ]
  [0.00079278 0.         0.02412169 ... 0.09786917 0.08757885
   0.        ]
  ...
  [0.00078232 0.         0.02817408 ... 0.1174852  0.10488396
   0.        ]
  [0.00076904 0.         0.02792925 ... 0.11649207 0.10398728
   0.        ]
  [0.00078886 0.         0.02519708 ... 0.07515864 0.09633785
   0.0154385 ]]

 [[0.00081761 0.02279603 0.00450244 ... 0.06548382 0.05917993
   0.        ]
  [0.00079905 0.         0.02374605 ... 0.09725887 0.08703511
   0.        ]
  [0.0008021  0.         0.02362086 ... 0.09767127 0.08715732
   0.        ]
  ...
  [0.00079264 0.         0.0274158  ... 0.11842685 0.10532099
   0.        ]
  [0.00078339 0.         0.02747761 ... 0.11711418 0.10436255
   0.        ]
  [0.00078195 0.         0.02509795 ... 0.07559878 0.09724852
   0.01529109]]

 [[0.00079893 0.0483374  0.00434778 ... 0.08793738 0.06925277
   0.        ]
  [0.00078635 0.02150546 0.0043228  ... 0.11018532 0.06228337
   0.        ]
  [0.00078263 0.02157461 0.00423865 ... 0.11068702 0.06296089
   0.        ]
  ...
  [0.00076916 0.02544353 0.00420289 ... 0.13430521 0.07541514
   0.        ]
  [0.00078152 0.02519465 0.00414102 ... 0.13276103 0.07484147
   0.        ]
  [0.00078309 0.         0.00756669 ... 0.09386374 0.05554575
   0.02451253]]]], shape=(64, 56, 56, 128), dtype=float32)
```

Out[ ]:
```
TensorShape([64, 56, 56, 128])
```

e) At several points in the model, two coordinate feature maps will be added to the output of a convolutional layer. This operation will be applied before each ResBlock and the classifier component of the CNN prediction model.

These two feature maps indicate relative x and y spatial position, and are each scaled from -1 to 1 across the height and width dimensions. These two feature maps are concatenated as two extra channels to the convolutional layer output.

Implement this operation as another custom layer class called `AddSpatialCoordinates`. The layer should be able to accept input Tensors with arbitrary height, width and channel dimensions. This custom layer will not have any trainable variables.

Create an instance of your custom layer class and test it on some dummy inputs to verify it works as expected.

(7 marks)

In [ ]:
```
class AddSpatialCoordinates(Layer):
    def __init__(self, name = "ASC"):
        super().__init__(name = name)

    def call(self,inputs):
        #here, inputs is the (batch,w,h,channel) tensor
        #[0] is batch size and [3] is number of channels
        x_size = inputs.shape[1]
        y_size = inputs.shape[2]

        x_grid = tf.linspace(-1,1,num = x_size)
        y_grid = tf.linspace(-1,1,num = y_size)
        #make these grids float32 to be compatible with the inputs tensor
        x_grid = tf.cast(x_grid,tf.float32)
        y_grid = tf.cast(y_grid,tf.float32)
        #this makes a list of two grids, -1 for x is left, -1 for y is top
        mesh = tf.meshgrid(x_grid,y_grid)
        #now we have to broadcast mesh so that we can add it to the inputs tenso
        mesh_x = mesh[0]
        mesh_y = mesh[1]
        mesh_x = tf.expand_dims(tf.expand_dims(mesh_x,0),-1)
        mesh_y = tf.expand_dims(tf.expand_dims(mesh_y,0),-1)
        #now tile to commute with the batch size
        mesh_x = tf.tile(mesh_x, [tf.shape(inputs)[0],1,1,1])
        mesh_y = tf.tile(mesh_y, [tf.shape(inputs)[0],1,1,1])
        inputs = tf.concat([inputs,mesh_x,mesh_y],axis = -1)
        return inputs
```

In [ ]:
```
#Again test this on the example input:
asc = AddSpatialCoordinates()
test = asc(cnn_feature_extractor(inputs[1]))
print(test.shape)
#this is giving what we expect, with the two added dimensions for channel
```

```
(64, 56, 56, 130)
```

In [ ]:
```
#lets inspect the final two dimensions of this, for the first example in the bat
print(test[0,:,:,-1])
print(test[0,:,:,-2])
#this is the relative positionings for each one as we eould expect
```

```
tf.Tensor(
[[-1.         -1.         -1.         ... -1.         -1.
  -1.        ]
 [-0.96363634 -0.96363634 -0.96363634 ... -0.96363634 -0.96363634
  -0.96363634]
 [-0.92727274 -0.92727274 -0.92727274 ... -0.92727274 -0.92727274
  -0.92727274]
 ...
 [ 0.92727274  0.92727274  0.92727274 ...  0.92727274  0.92727274
   0.92727274]
 [ 0.96363634  0.96363634  0.96363634 ...  0.96363634  0.96363634
   0.96363634]
 [ 1.          1.          1.         ...  1.          1.
   1.        ]], shape=(56, 56), dtype=float32)
tf.Tensor(
[[-1.         -0.96363634 -0.92727274 ...  0.92727274  0.96363634
   1.        ]
 [-1.         -0.96363634 -0.92727274 ...  0.92727274  0.96363634
   1.        ]
 [-1.         -0.96363634 -0.92727274 ...  0.92727274  0.96363634
   1.        ]
 ...
 [-1.         -0.96363634 -0.92727274 ...  0.92727274  0.96363634
   1.        ]
 [-1.         -0.96363634 -0.92727274 ...  0.92727274  0.96363634
   1.        ]
 [-1.         -0.96363634 -0.92727274 ...  0.92727274  0.96363634
   1.        ]], shape=(56, 56), dtype=float32)
```

f) The final main block of the CNN network is a classifier block. This block consists of the following layers:

- 1x1 convolution with 512 output channels, ReLU activation, and 'SAME' padding
- Global max pooling across height and width dimensions
- Dense layer with 512 neurons and ReLU activation
- Final Dense layer with n_c neurons and softmax activation, where n_c is the number of output labels

Once you have implemented the classifier, you should bring all components together to build the complete model. This model consists of the following:

- GRU FiLM generator as defined in part a) that processes the sequence of question tokens and outputs an embedding $\mathbf{q}$ of dimension 128
- Feature extractor block as defined in part b) that processes the input image
- The output of the feature extractor should then be extended with spatial coordinate feature maps by passing it through your `AddSpatialCoordinates` layer
- This should be followed by just one ResBlock custom layer, that takes in two inputs: the output from the previous `AddSpatialCoordinates` layer and the question embedding $\mathbf{q}$. We will only use one ResBlock due to computational limitations
- The output of the ResBlock should then also be extended with spatial coordinate feature maps by passing it through your `AddSpatialCoordinates` layer
- The output from the previous `AddSpatialCoordinates` layer should then be sent through the classifier block to obtain the final output prediction

Implement the complete model according to the above spec, and print the model summary.

(5 marks)

In [ ]:
```
def get_classifier():
    classifier = Sequential([
        #dont need to specify input size for conv2d
        Conv2D(512,(1,1), activation = "relu", padding = "same"),
        #looking at documentation, this globally pools over height and width, le
        GlobalMaxPooling2D(),
        Dense(512,activation = "relu"),
        Dense(vocab_ans_size, activation = "softmax")], name = "classifier")
    return classifier
```

In [ ]:
```
class full_model(Model):
    def __init__(self):
        super(full_model,self).__init__()
        self.gru = get_gru()
        self.feature_extractor = get_cnn_feature_extractor()
        #asp can be called multiple times since not trained
        self.asp = AddSpatialCoordinates()
        self.resblock = ResBlock()
        self.classifier = get_classifier()
    def call(self,inputs):
        #we expect just the input to be passed in, not the target
        text, images = inputs
        q = self.gru(text)
        h = self.feature_extractor(images)
        h = self.asp(h)
        h = self.resblock(q,h)
        h = self.asp(h)
        #predict
        preds = self.classifier(h)
        return preds
```

In [ ]:
```
#to print the model summary we call it on the example
fm = full_model()
print(fm(inputs))
fm(inputs).shape
#we expected this shape, since the batch is 64 and there were 30 possible answer
#and each of the 30 have a roughly equal chance, which makes sense since the mod
```

```
tf.Tensor(
[[0.03206209 0.03023569 0.04686745 ... 0.03364512 0.0383742  0.02747775]
 [0.0320677  0.03036618 0.0469934  ... 0.03357996 0.03850182 0.02734053]
 [0.0320782  0.03024117 0.0470595  ... 0.03358467 0.03857011 0.02739289]
 ...
 [0.03203578 0.03035682 0.04688934 ... 0.03358771 0.03846231 0.02732208]
 [0.03208403 0.03027789 0.04689825 ... 0.03359709 0.03853745 0.02747268]
 [0.03201549 0.03013543 0.04685825 ... 0.03358852 0.0384712  0.02738458]], shape=
(64, 30), dtype=float32)
```

Out[ ]:
```
TensorShape([64, 30])
```

In [ ]:
```
fm.summary()
```

```
Model: "full_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 gru_network (Sequential)    (None, 128)               178816    
                                                                 
 feature_extractor (Sequenti  (None, 56, 56, 128)      269568    
 al)                                                             
                                                                 
 ASC (AddSpatialCoordinates)  multiple                 0         
                                                                 
 res_block_1 (ResBlock)      multiple                  197376    
                                                                 
 classifier (Sequential)     (64, 30)                  345118    
                                                                 
=================================================================
Total params: 990,878
Trainable params: 990,110
Non-trainable params: 768
_________________________________________________________________
```


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 gru_network (Sequential)    (None, 128)               178816    
                                                                 
 feature_extractor (Sequenti  (None, 56, 56, 128)      269568    
 al)                                                             
                                                                 
 ASC (AddSpatialCoordinates)  multiple                 0         
                                                                 
 res_block_1 (ResBlock)      multiple                  197376    
                                                                 
 classifier (Sequential)     (64, 30)                  345118    
                                                                 
=================================================================
Total params: 990,878
Trainable params: 990,110
Non-trainable params: 768
_________________________________________________________________
```


Question 3 (Total 30 marks)

a) You should now train your model from question 2 using a cross entropy loss function. Train the model for 20 epochs, with an Adam optimizer with learning rate 3e-4. You should track model performance on the validation set, including the accuracy.

Your code should be structured to account for restarting broken training runs. You will need to save your model every epoch, and save all of the model's training and validation performance up to that point (a convenient method is to use the `CSVLogger` callback). In the case of a broken training run, the required data should be loaded, and the training run resumed from the last saved checkpoint. You do not need to use early stopping in the training run.

When training has completed, compute and print the final evaluation of your model on the validation set.

*NB: The model would need to be larger and trained for longer to achieve good performance on this task. The model and training have been scaled down to accommodate infrastructure limitations on the Coursera platform. You should implement the architecture as specified in this assessment, but you can train the model for longer if you wish. The performance of the resulting model is **not** part of the marking criteria.*

(15 marks)

In [ ]:
```
#setup optimizer and callback
optimizer = Adam(lr = 3e-4)
# callback to save training and validation performance in case of broken runs
logger = CSVLogger("epoch_log.csv", separator = ",", append = True)
# callback to save the model every epoch, but only record the best to avoid havi
#model performance is to be tracked on the validation loss, but still include ac
ckpt = ModelCheckpoint("best_model.h5", monitor = "val_loss", save_best_only= Tr
fm.compile(loss = "SparseCategoricalCrossentropy", optimizer = optimizer,metrics
```

```
d:\University-local\Imperial\Term 2\Deep Learning\.conda\lib\site-packages\keras
\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr` argument is deprecate
d, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

In [ ]:
```
fm_history = fm.fit(train_ds,epochs = 20, validation_data = val_ds, callbacks =
```

```
Epoch 1/20
```

```
234/234 [==============================] - 48s 169ms/step - loss: 2.0546 - accura
cy: 0.2856 - val_loss: 1.9809 - val_accuracy: 0.3432
Epoch 2/20
234/234 [==============================] - 36s 153ms/step - loss: 1.3256 - accura
cy: 0.3791 - val_loss: 1.2824 - val_accuracy: 0.4155
Epoch 3/20
234/234 [==============================] - 36s 154ms/step - loss: 1.0969 - accura
cy: 0.4296 - val_loss: 1.0822 - val_accuracy: 0.4275
Epoch 4/20
234/234 [==============================] - 38s 163ms/step - loss: 1.0667 - accura
cy: 0.4258 - val_loss: 1.0573 - val_accuracy: 0.4429
Epoch 5/20
234/234 [==============================] - 39s 164ms/step - loss: 1.0480 - accura
cy: 0.4372 - val_loss: 1.0134 - val_accuracy: 0.4555
Epoch 6/20
234/234 [==============================] - 35s 149ms/step - loss: 1.0276 - accura
cy: 0.4405 - val_loss: 1.0190 - val_accuracy: 0.4411
Epoch 7/20
234/234 [==============================] - 35s 150ms/step - loss: 1.0143 - accura
cy: 0.4415 - val_loss: 1.0182 - val_accuracy: 0.4413
Epoch 8/20
234/234 [==============================] - 35s 150ms/step - loss: 0.9994 - accura
cy: 0.4413 - val_loss: 0.9999 - val_accuracy: 0.4520
Epoch 9/20
234/234 [==============================] - 35s 150ms/step - loss: 1.0027 - accura
cy: 0.4478 - val_loss: 1.0121 - val_accuracy: 0.4389
Epoch 10/20
234/234 [==============================] - 35s 150ms/step - loss: 1.0005 - accura
cy: 0.4401 - val_loss: 0.9881 - val_accuracy: 0.4501
Epoch 11/20
234/234 [==============================] - 35s 151ms/step - loss: 0.9781 - accura
cy: 0.4441 - val_loss: 0.9858 - val_accuracy: 0.4317
Epoch 12/20
234/234 [==============================] - 36s 151ms/step - loss: 0.9751 - accura
cy: 0.4463 - val_loss: 0.9721 - val_accuracy: 0.4344
Epoch 13/20
234/234 [==============================] - 35s 150ms/step - loss: 0.9737 - accura
cy: 0.4477 - val_loss: 0.9879 - val_accuracy: 0.4397
Epoch 14/20
234/234 [==============================] - 35s 150ms/step - loss: 0.9577 - accura
cy: 0.4529 - val_loss: 0.9698 - val_accuracy: 0.4320
Epoch 15/20
234/234 [==============================] - 35s 150ms/step - loss: 0.9633 - accura
cy: 0.4522 - val_loss: 0.9626 - val_accuracy: 0.4357
Epoch 16/20
234/234 [==============================] - 35s 150ms/step - loss: 0.9573 - accura
cy: 0.4556 - val_loss: 0.9678 - val_accuracy: 0.4429
Epoch 17/20
234/234 [==============================] - 36s 154ms/step - loss: 0.9557 - accura
cy: 0.4539 - val_loss: 0.9570 - val_accuracy: 0.4597
Epoch 18/20
234/234 [==============================] - 38s 162ms/step - loss: 0.9581 - accura
cy: 0.4530 - val_loss: 0.9570 - val_accuracy: 0.4584
Epoch 19/20
234/234 [==============================] - 38s 163ms/step - loss: 0.9444 - accura
cy: 0.4686 - val_loss: 0.9408 - val_accuracy: 0.4587
Epoch 20/20
234/234 [==============================] - 36s 153ms/step - loss: 0.9526 - accura
cy: 0.4532 - val_loss: 0.9594 - val_accuracy: 0.4648
```

In [ ]:
```
# I had issues where if I saved the entire best model and then Loaded it, on a f


#test the best model on the validation set (as stated on ED)
#we do so by Loading from the file, to ensure it can be rexecuted if needed
best_model = full_model()
#initialize
best_model(inputs)
#compile and Load weights
best_model.compile(loss = "SparseCategoricalCrossentropy", optimizer = optimizer
best_model.load_weights("best_model.h5")
best_model.evaluate(val_ds)
```

```
59/59 [==============================] - 7s 60ms/step - loss: 0.9377 - accuracy:
0.4587
```

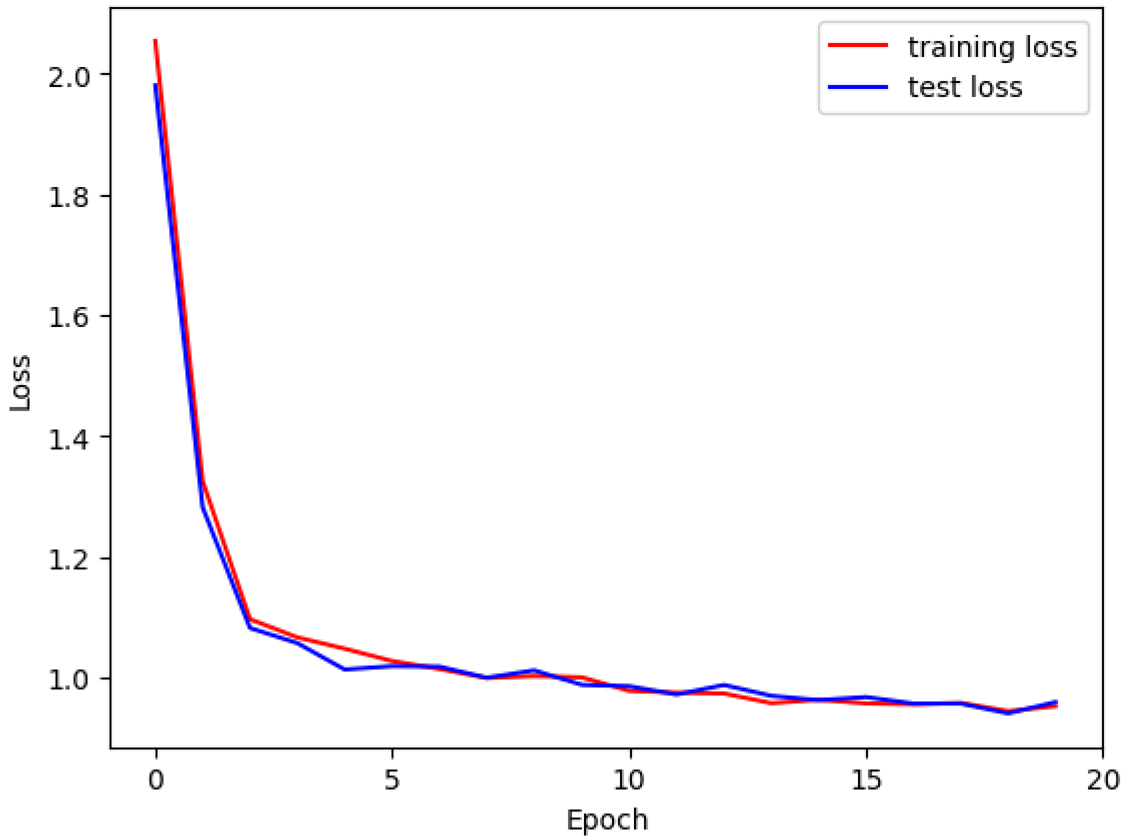
Out[ ]:
```
[0.9377139806747437, 0.4586666524410248]
```

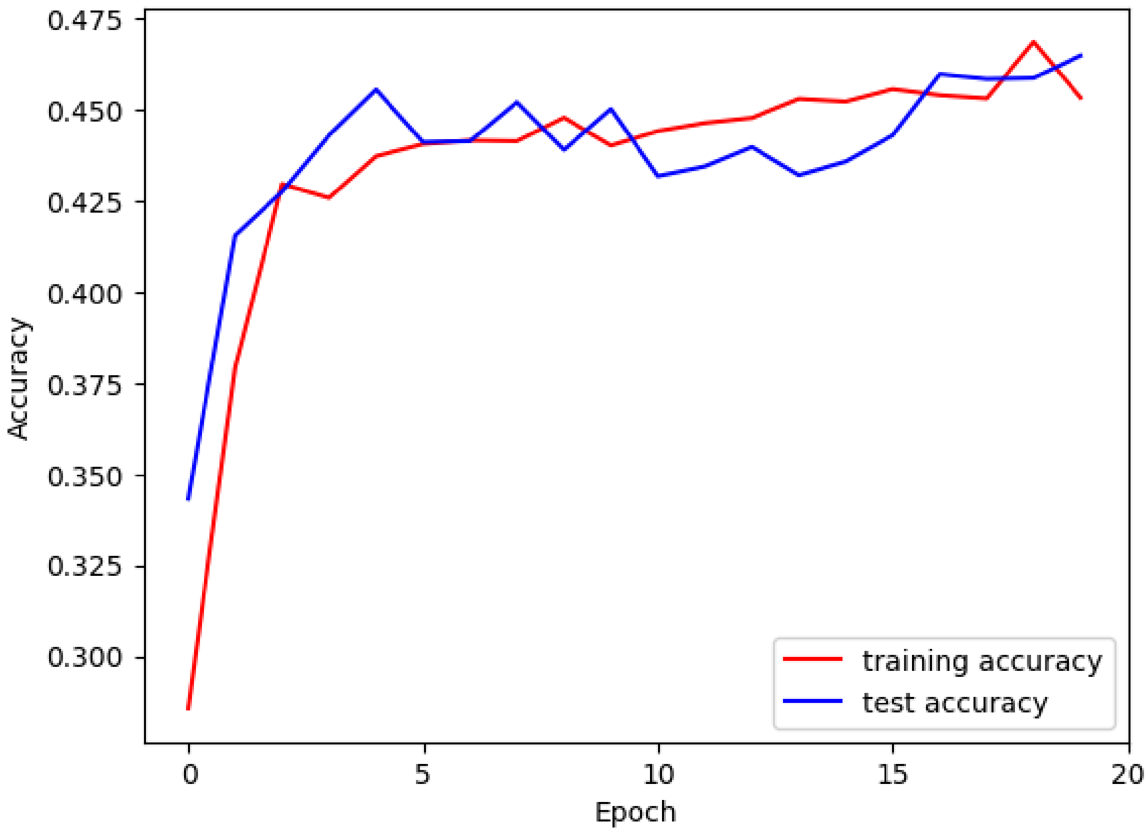
b) Plot the loss and accuracy over the course of training on the training and validation sets.

Select at least one sample image and question from the validation set, and compute the model predictions. Display the image, question, ground truth answer and model predictive distribution over the set of answers.

(7 marks)

In [ ]:
```
plt.plot(fm_history.history["loss"], "r", label = "training loss")
plt.plot(fm_history.history["val_loss"],"b",label = "test loss")
plt.xlabel("Epoch")
plt.xticks([0,5,10,15,20])
plt.ylabel("Loss")
plt.legend()
plt.show()

plt.plot(fm_history.history["accuracy"], "r", label = "training accuracy")
plt.plot(fm_history.history["val_accuracy"],"b",label = "test accuracy")
plt.xlabel("Epoch")
plt.xticks([0,5,10,15,20])
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

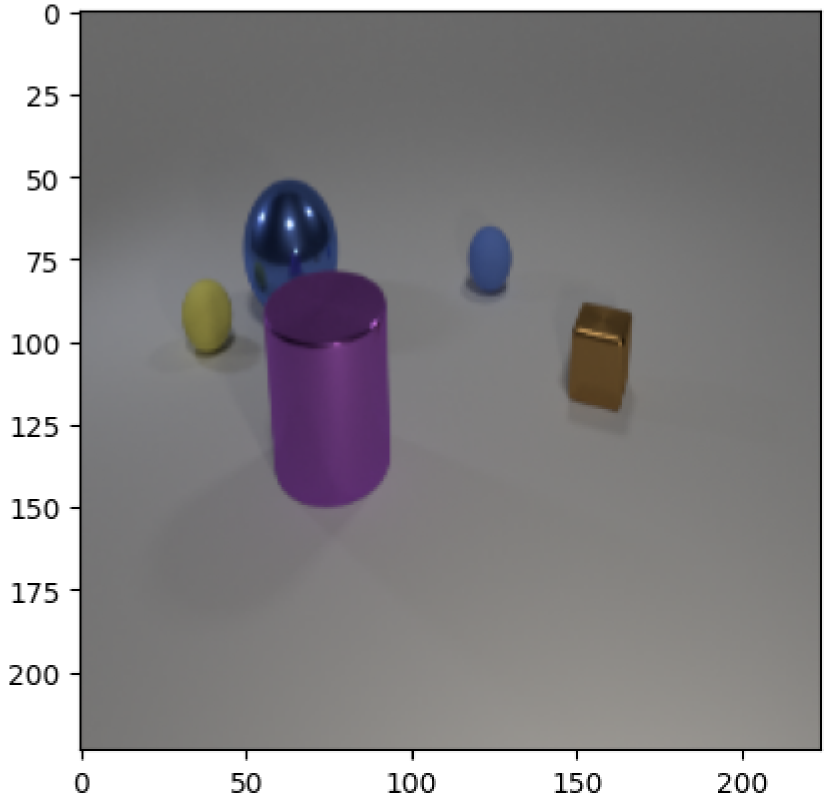




Next, we pick a random question from the validation set and predict the response:

In [ ]:
```
#select a batch
for ele in val_ds.take(1):
    eg, eg_a= ele[0], ele[1]
#randomly select an item from the batch
r = np.random.randint(0,eg_a.shape[0]-1)
```

```
eg_q,eg_i= eg[0][r], eg[1][r]
eg_a = eg_a[r]
plt.imshow(eg_i)
plt.show()

#print question
eg_q_words = [vocab_q[idx] for idx in eg_q if idx != 0]
eg_q_words[0] = eg_q_words[0].capitalize()
eg_q_sentence = ' '.join(word for word in eg_q_words)
print(f"Question: {eg_q_sentence}?")
print(f"Answer: {vocab_ans[eg_a]}")
```



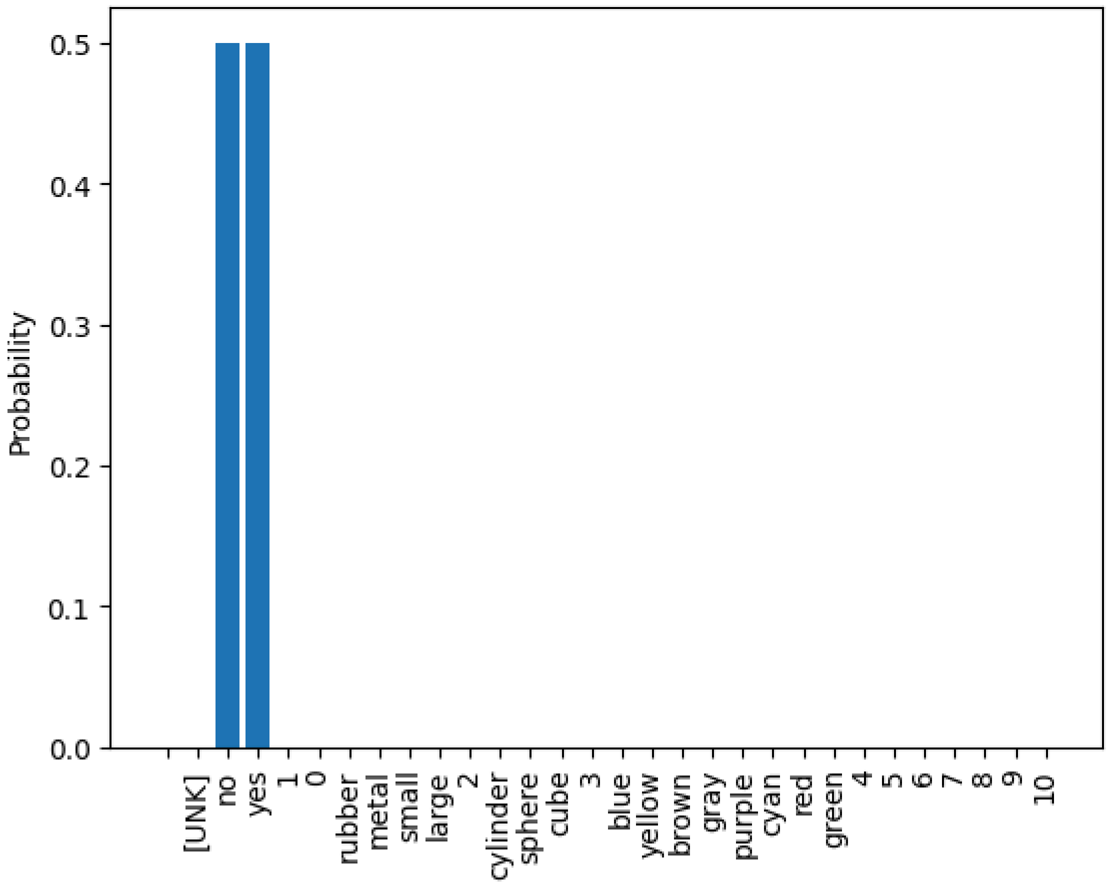

```
Question: Do the yellow sphere and the metal object that is behind the metallic b
lock have the same size?
Answer: no
```

Now, for the predicted response, and predictive categorical distribution:

In [ ]:
```
pred = fm.predict((eg_q[None,:],eg_i[None,:])).squeeze(0)
print(f"Predicted answer: {vocab_ans[tf.argmax(pred)]}")
```

```
1/1 [==============================] - 2s 2s/step
Predicted answer: no
1/1 [==============================] - 2s 2s/step
Predicted answer: no
```

In [ ]:
```
plt.bar(vocab_ans, pred)
plt.xticks(rotation=90)
plt.ylabel("Probability")
plt.show()
```



In [ ]:
```
#predictive distribution as a table, we do as 3 in order to fully show
table = [[vocab_ans[i],pred[i]] for i in range(10)]
print(tabulate(table, headers = ["Response","Probability"],tablefmt = "fancy_gri
```

Response	Probability
	3.05033e-08
[UNK]	2.76312e-08
no	0.500041
yes	0.499763
1	3.23345e-05
0	2.59166e-05
rubber	5.74105e-07
metal	1.0419e-06
small	2.05424e-05
large	1.5781e-05

In [ ]:
```
table = [[vocab_ans[i],pred[i]] for i in range(10,20)]
print(tabulate(table, headers = ["Response","Probability"],tablefmt = "fancy_gri
```

Response	Probability
2	1.78211e-05
cylinder	2.03542e-06
sphere	4.24522e-06
cube	2.26384e-06
3	9.57441e-06
blue	6.6474e-06
yellow	7.50079e-06
brown	8.13632e-06
gray	5.20654e-06
purple	9.6281e-06

In [ ]:
```
table = [[vocab_ans[i],pred[i]] for i in range(20,30)]
print(tabulate(table, headers = ["Response","Probability"],tablefmt = "fancy_gri
```

Response	Probability
cyan	6.34695e-06
red	6.42977e-06
green	3.78207e-06
4	5.58822e-06
5	2.89542e-06
6	6.24316e-07
7	4.83194e-07
8	1.15183e-07
9	1.51027e-07
10	2.55918e-08

c) Explain why adding spatial coordinate feature maps as in 2e) is beneficial for the proposed model and task from questions 1 and 2.

(3 marks)

It is beneficial because in the context of the questions asked about each image, relative positions of the different objects are crucial to answering the questions correctly. In usual image detection, the CNN can simply learn edges and then other features, without caring too much about where different objects in the image are in relation to each other, since they were made with translation invariance in mind. However, when the learning task requires these relative positionings within the image, the spacial coordinate feature maps are incredibly helpful, since they provide the model with the local information needed so it can learn relative positionings in relation to the words "left", "right", "behind" etc. It also helps with differentiating between two similar looking objects that are located within different parts of the image (which is very important in the context of the dataset questions).

d) An alternative method to FiLM to incorporate conditioning information would be to concatenate the conditional embedding $\mathbf{q}$ with the channel dimension in a convolutional layer input $\mathbf{h}^{(k-1)}$ at every spatial location (in other words, concatenate constant feature maps with the input $\mathbf{h}^{(k-1)}$ to a convolutional layer).

Explain how this method would compare in terms of computation and parameter efficiency with applying the FiLM layer computation outlined in 2c) to the output $\mathbf{h}^{(k)}$ of the convolutional layer. You can assume the convolutional layer has no activation function.

(5 marks)

The FiLM layer shifts and scales the prior output from the batch normalisation layer coming from the CNN part of the block. The shift and scale are aquired via two separate linear layers with the signal q from the GRU part of the network. These linear layers would have parameters that have to be learned within the FiLM layer. The alternative method concatenates q with h before h goes through the convolutional layer. These two methods would have different numbers of parameters, and different computational costs.

Firstly, increasing the channel dimension of h by concatenating q is going to increase the computational time, especially considering this is a convolutional layer. Concatenating the conditional embedding at every spatial location means the shape of the tensor would go from (H,W,C) to (H,W,C+D) where C is the number of channels h has before, and D is the dimension of the conditional embedding. In our case, this means an increase in channel dimension from 128 to 256. It comes down to a comparison of the computations required for doubling the channel dimension going into the convolutional layer, and the FiLM layer. In general, considering the kernel shape can be increased and changed, this would lead to more computations required, than the pretty much fixed number required for the FiLM layer. Thus, in general, FiLM is more efficient computationally.

As for parameter efficiency, the FiLM layer introduces a γ and a β for each channel of the incoming output $\mathbf{h}^{(k)}$ of the convolutional layer. This means, assuming a fully connected linear layer for each (with no biases), there would be $128^2 \times 2 = 32768$ parameters to be learned using the FiLM layer. However, if instead the second method was used, concatenating q and h at every spacial location before the convolutional layer, this would result in a channel dimension of $128 + 128 = 256$. Now the number of parameters would depend on the convolutional layer used, but for arguments sake, lets take the first convolutional layer in the resblock. This has 128 filters and a kernel size of 1x1. Then we are comparing inputs with the same height, width, and batch dimensions, but differet numbers of channels. The FiLM layer method would have 128 channels at this point, resulting in $128 \times 128 + 128 = 16512$ parameters to be learned (including bias here), whereas the alternative method would have 256 channels, resulting in $256 \times 128 + 128 = 32,896$ parameters to be learned (again including bias). However, the FiLM layer has the additional parameters to be learned from the layer itself, resulting in 49280 parameters to learned in total, meaning a difference of 16384 parameters between the two methods. This was done for the first convolutional layer in the resblock, but generally, using the alternative doubles the number of parameters (before including the bias terms) in the convolutional layer, but doesn't have the parameters to be learned in the FiLM layer. However, the number of parameters to be learned in the FiLM layer can be considered fixed, assuming the input channels remains constant. If the kernel size in the convolutional layer is increased, then then the number of parameters needed to be learned is increased, thus when the second method is used and this number is then doubled, it can lead to a much greater total number of parameters for the second method, compared to the first, even with the 32768 less from not having the FiLM parameters. This can be seen below:

In [ ]:
```
#Setup dummy inputs
for ele in train_ds.take(1):
    inputs, outputs = ele

q_test = gru_network(inputs[0])
h_test = cnn_feature_extractor(inputs[1])

#for FiLM layer, this is done after the Conv layer, so q would come later, but w
q_test = q_test[:,None,None,:]
q_test = tf.tile(q_test,[1,56,56,1])
h_test_alternative = tf.concat([h_test,q_test],axis = -1)
print(h_test_alternative.shape)
```

```
(64, 56, 56, 256)
```

In [ ]:
```
#summary for
test_layer= Sequential([
    Conv2D(128, (1,1), activation = "relu", padding = "same")
])
test_layer(h_test)
print(test_layer.summary())

test_layer= Sequential([
    Conv2D(128, (1,1), activation = "relu", padding = "same")
])
test_layer(h_test_alternative)
print("---"*100, "\n")
print("ALTERATIVE METHOD:")
print(test_layer.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (64, 56, 56, 128)         16512

=================================================================
Total params: 16,512
Trainable params: 16,512
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (64, 56, 56, 128)         16512

=================================================================
Total params: 16,512
Trainable params: 16,512
Non-trainable params: 0
_________________________________________________________________
None
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------------------------------------

ALTERATIVE METHOD:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (64, 56, 56, 128)         32896

=================================================================
Total params: 32,896
Trainable params: 32,896
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:
```
# NOW FOR A BIGGER KERNEL
#summary for
test_layer= Sequential([
    Conv2D(128, 10, activation = "relu", padding = "same")
])
test_layer(h_test)
print(test_layer.summary())

test_layer= Sequential([
    Conv2D(128, 10, activation = "relu", padding = "same")
])
test_layer(h_test_alternative)
print("---"*100, "\n")
print("ALTERATIVE METHOD:")
print(test_layer.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (64, 56, 56, 128)         1638528

=================================================================
Total params: 1,638,528
Trainable params: 1,638,528
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (64, 56, 56, 128)         1638528

=================================================================
Total params: 1,638,528
Trainable params: 1,638,528
Non-trainable params: 0
_________________________________________________________________
None
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
--------------------------------------------------------

ALTERATIVE METHOD:
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (64, 56, 56, 128)         3276928

=================================================================
Total params: 3,276,928
Trainable params: 3,276,928
Non-trainable params: 0
_________________________________________________________________
None
```

Hence again, in general, the FiLM layer is going to be more efficient with respect to the number of parameters needed.

Question 4 (Total 10 marks)

Provide a separate PDF report with your evaluation and conclusions on the model and training results in this assessment.

In addition, compare the experiment conducted in this assessment with that described in section 2 of the original paper. In particular, discuss how the model architecture and training algorithm differ.

Your report should be no more than 1 page.

(10 marks)