Dataset Setup

In [ ]:

```
import pathlib, os, sys, operator, re, datetime
from functools import reduce
import numpy as np
import tensorflow as tf
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import Model
import tensorflow_datasets as tfds
from tiny_imagenet import TinyImagenetDataset
from keras.models import load_model

# Enable or disable GPU
# To fully disable it, we need to hide all GPU devices from Tensorflow
# Make sure GPU is disabled for this inference part of the lab
ENABLE_GPU = True
# tf.debugging.set_log_device_placement(True)

if not ENABLE_GPU:
    tf.config.set_visible_devices([], 'GPU')

# Print Python and TF version, and where we are running
print(f'Running on Python Version: {sys.version}')
print(f'Using Tensorflow Version: {tf. __version__}')
if not tf.config.experimental.list_physical_devices("GPU"):
    print('Running on CPU')
else:
    print(f'Using GPU at: {tf.test.gpu_device_name()} (of {len(tf.config.expe
```

```
Running on Python Version: 3.6.8 (default, May 31 2023, 10:28:59)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-18)]
Using Tensorflow Version: 2.6.2
Using GPU at: /device:GPU:0 (of 1 available)
```

In [ ]:

```
# Original Source: https://github.com/ksachdeva/tiny-imagenet-tfds
# Class Version Source: https://github.com/Mluckydwyer/tiny-imagenet-tfds
# Setup our dataset
# ---------------------------------------------------------

tiny_imagenet_builder = TinyImagenetDataset()

# this call (download_and_prepare) will trigger the download of the dataset
# and preparation (conversion to tfrecords)
#
# This will be done only once and on next usage tfds will
# use the cached version on your host.
tiny_imagenet_builder.download_and_prepare(download_dir="~/tensorflow-dataset

# class_names = tiny_imagenet_builder.info.features['label'].names
ds = tiny_imagenet_builder.as_dataset()
ds_train, ds_val = ds["train"], ds["validation"]
assert(isinstance(ds_train, tf.data.Dataset))
assert(isinstance(ds_val, tf.data.Dataset))

# Training Dataset
ds_train = ds_train.shuffle(1024).prefetch(tf.data.AUTOTUNE)

# Validation Dataset
ds_val = ds_val.shuffle(1024).prefetch(tf.data.AUTOTUNE)

# Dataset metadata
ds_info = tiny_imagenet_builder.info
```

Working with the Dataset

In [ ]:

```
# We need to read the "human readable" labels so we can translate with the nu
# Read the labels file (words.txt)
with open(os.path.abspath('wnids.txt'), 'r') as f:
    wnids = [x.strip() for x in f]

# Map wnids to integer labels
wnid_to_label = {wnid: i for i, wnid in enumerate(wnids)}
label_to_wnid = {v: k for k, v in wnid_to_label.items()}

# Use words.txt to get names for each class
with open(os.path.abspath('words.txt'), 'r') as f:
    wnid_to_words = dict(line.split('\t') for line in f)
    for wnid, words in wnid_to_words.items():
        wnid_to_words[wnid] = [w.strip() for w in words.split(',')]

class_names = [str(wnid_to_words[wnid]) for wnid in wnids]
```

In [ ]:

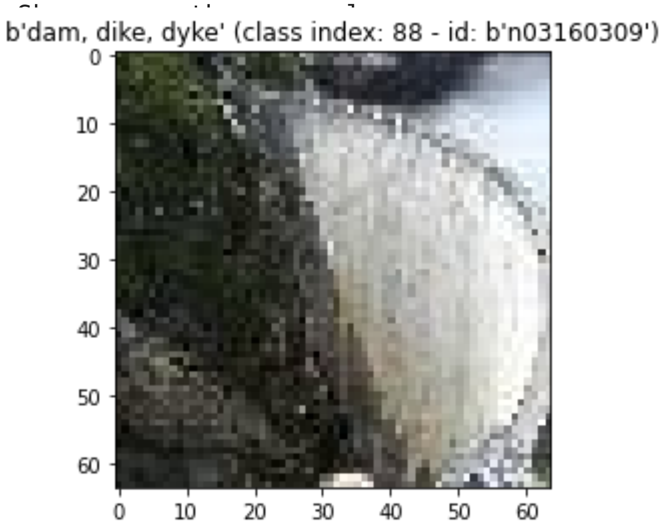
```
# Helper function to get the label name
def img_class(img_data, idx=None):
    image, label, id, label_name = img_data["image"], img_data["label"], img_
    # Handle batches of images correctly
    if idx != None:
        image, label, id, label_name = img_data["image"][idx], img_data["labe

    return f"{label_name} (class index: {label} - id: {id})"


# Helper function to show basic info about an image
def img_info(img, idx=None, display=True, title_apend=""):
    image = img['image']

    # Print the class
    class_str = img_class(img, idx)
    print(f"Label: {class_str}")

    # Display the image
    if display:
        plt.figure()
        plt.title(title_apend + class_str)
        # Handle batches correctly
        if image.shape.ndims > 3:
            plt.imshow(image.numpy().reshape(64, 64, 3))
        else:
            plt.imshow(image.numpy())
```

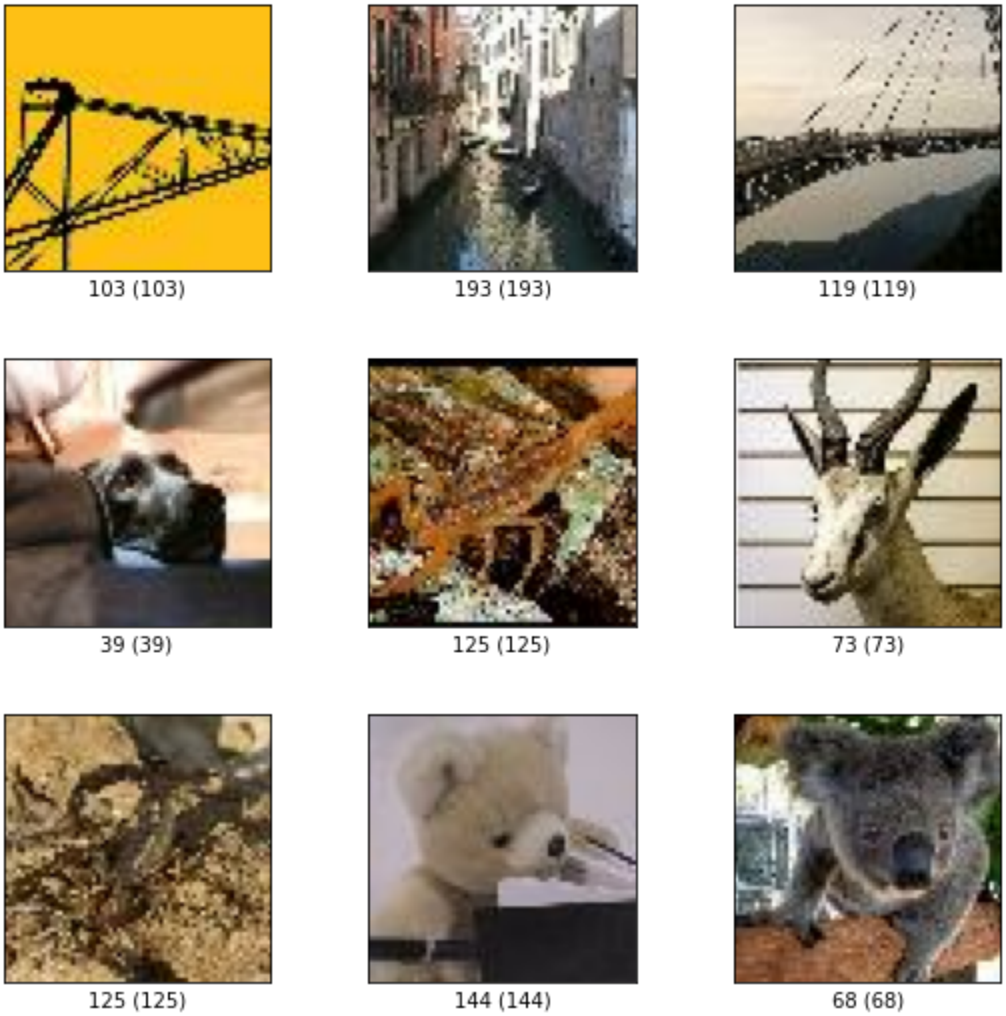
In [ ]:

```
# Print the dataset types and info
print("--- Train & Validation dataset info ---")
print(f"Train: {ds_train}")
print(f"Validation: {ds_val}")
print(f"Dataset Info: {ds_info}") # Uncomment to print Dataset info

print("\n--- Show an example image ---")
for example in ds_val.take(1):
    img_info(example)

print("\n Show some other examples")
tfds.show_examples(ds_val, ds_info, rows=3, cols=3)
```

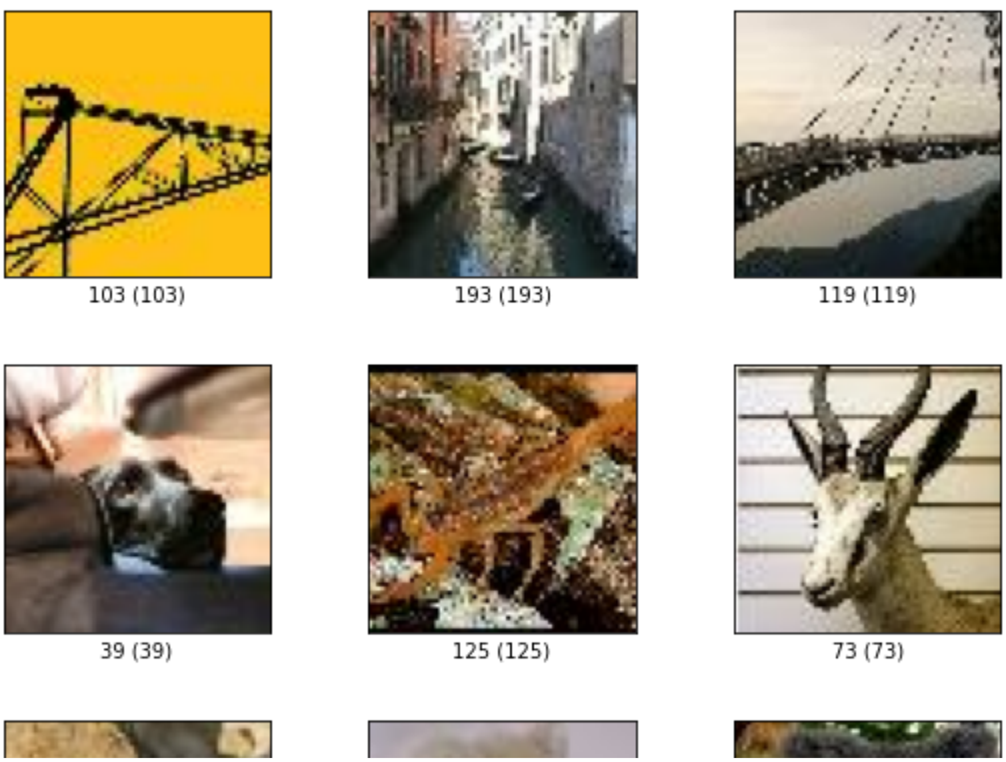
```
--- Train & Validation dataset info ---
Train: <PrefetchDataset shapes: {id: (), image: (64, 64, 3), label: (), metad
ata: {label_name: ()}}, types: {id: tf.string, image: tf.uint8, label: tf.int
64, metadata: {label_name: tf.string}}>
Validation: <PrefetchDataset shapes: {id: (), image: (64, 64, 3), label: (),
metadata: {label_name: ()}}, types: {id: tf.string, image: tf.uint8, label: t
f.int64, metadata: {label_name: tf.string}}>
Dataset Info: tfds.core.DatasetInfo(
    name='tiny_imagenet_dataset',
    full_name='tiny_imagenet_dataset/0.2.0',
    description="""
    Tiny ImageNet Challenge is a similar challenge as ImageNet with a smaller
dataset but
                             less image classes. It contains 200 image classe
s, a training
```

```
                             dataset of 100, 000 images, a validation dataset
of 10, 000
                             images, and a test dataset of 10, 000 images. Al
l images are
                             of size 64×64.
    """,
    homepage='https://www.tensorflow.org/datasets/catalog/tiny_imagenet_datas
et',
    data_path='/home/tjfriedl/tensorflow_datasets/tiny_imagenet_dataset/0.2.0
',
    download_size=236.61 MiB,
    dataset_size=215.40 MiB,
    features=FeaturesDict({
        'id': Text(shape=(), dtype=tf.string),
        'image': Image(shape=(64, 64, 3), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=200),
        'metadata': FeaturesDict({
            'label_name': tf.string,
        }),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'train': <SplitInfo num_examples=100000, num_shards=2>,
        'validation': <SplitInfo num_examples=10000, num_shards=1>,
    },
    citation="""@article{tiny-imagenet,
                                  author = {Li,Fei-Fei}, {Karpathy,Andrej} an
d {Johnson,Justin}"}""",
)

--- Show an example image ---
Label: b'dam, dike, dyke' (class index: 88 - id: b'n03160309')
```




103 (103)
193 (193)
119 (119)
39 (39)
125 (125)
73 (73)
125 (125)
144 (144)
68 (68)

Out[ ]:

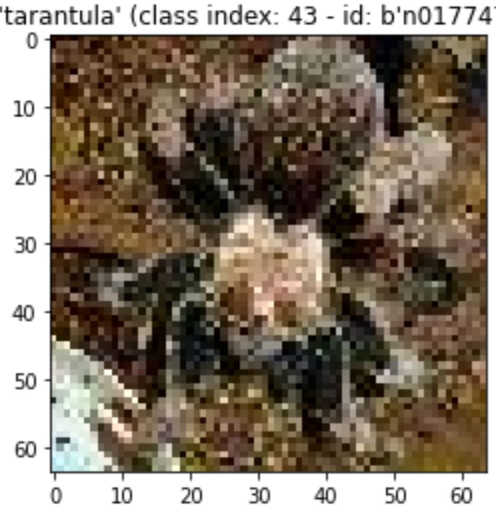

103 (103)
193 (193)
119 (119)
39 (39)
125 (125)
73 (73)

In [ ]:

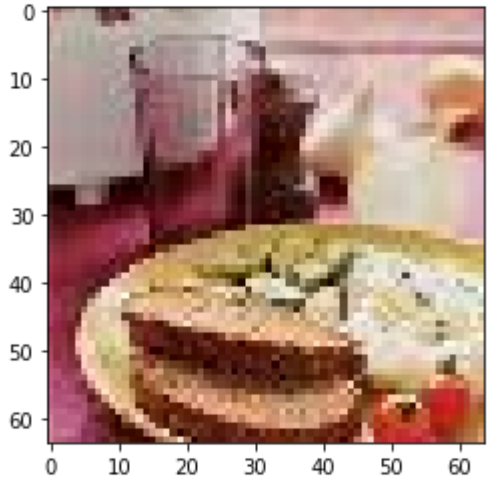
```
# TODO: Print and visualize three inputs from the validation set
#      : Print the storage data type
#      : Print and note the dimensions of each image
#      : Print the memory required to store each image

# Sample Images
sample_imgs = []
for index, img_data in enumerate(ds_val.take(3)):
    sample_imgs.append(img_data)
    image, label, id, label_name = img_data["image"], img_data["label"], img_

    data_type = image.dtype # Supposed storage data type
    image_dimensions = image.shape # Supposed image shape
    dtype_size = image.dtype.size
    mem_per_element = image_dimensions.num_elements()
    memory_required = dtype_size * mem_per_element

    print(f'\n--- Image {index} ---')
    print(f"Storage Data Type: {data_type}") # Print out the storage data typ
    print(f"Image Dimensions: {image_dimensions}") # Print and note the dimen
    print(f"Memory required: {memory_required} (Bytes)") # Print the memory r
    img_info(img_data)
```

```
--- Image 0 ---
Storage Data Type: <dtype: 'uint8'>
Image Dimensions: (64, 64, 3)
Memory required: 12288 (Bytes)
Label: b'tarantula' (class index: 43 - id: b'n01774750')

--- Image 1 ---
Storage Data Type: <dtype: 'uint8'>
Image Dimensions: (64, 64, 3)
Memory required: 12288 (Bytes)
Label: b'meat loaf, meatloaf' (class index: 192 - id: b'n07871810')

--- Image 2 ---
Storage Data Type: <dtype: 'uint8'>
Image Dimensions: (64, 64, 3)
Memory required: 12288 (Bytes)
Label: b'sea slug, nudibranch' (class index: 165 - id: b'n01950731')
```

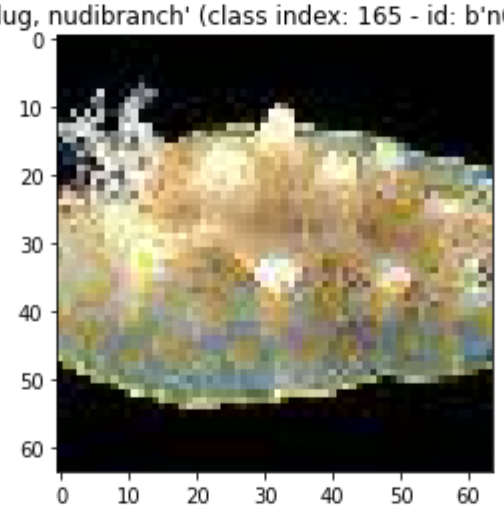
b'tarantula' (class index: 43 - id: b'n01774750')



b'meat loaf, meatloaf' (class index: 192 - id: b'n07871810')



b'sea slug, nudibranch' (class index: 165 - id: b'n01950731')



In [ ]:

```
# TODO: Export each of the three inputs to a binary file which will be used t
# NOTE: First flatten the array (ex: 4D --> 1D). So 64*64*3 = 12288 element 1

# Make a directory for our image data
img_dir = os.path.abspath('img_data')
pathlib.Path(img_dir).mkdir(exist_ok=True)

# Create a metadata file
metadata_file = open(os.path.join(img_dir, f'metadata.txt'), 'w')
metadata_file.write(f'Number\t\tDims\t\tClass Data\n')

# Export each image
for index, img_data in enumerate(sample_imgs):
    img_file = open(os.path.join(img_dir, f'image_{index}.bin'), 'wb')

    # TODO: Your Code Here
    image_data = img_data["image"].numpy() # Grabs the numerical image data f
    stringed_image_data = image_data.flatten() # Flattens the 4D array into b
    img_file.write(stringed_image_data.tobytes()) # Writes to the file in pro

    # print(stringed_image_data.shape) # Used to test we are acquiring 12288

    img_file.close()

    # Write the image metadata for reference later
    class_str = img_class(img_data)
    metadata_file.write(f'{index}\t\t{img_data["image"].shape}\t\t{class_str}
metadata_file.close()
```

Model Setup

In [ ]:

```
# TODO: Load the model
# Now we will load the H5 model! Please make sure the h5 model file is presen
# You can download this from the Canvas Page and place it in the same directo

# model_path = os.path.abspath(""/home/dwyer/482/dev/CNN_TinyImageNet_2.h5)"
model_path = os.path.abspath("CNN_TinyImageNet_2.h5")

# TODO: Your Code Here
model = tf.keras.models.load_model(model_path) # I believe this should do the

# TODO: Print a summary of the model
print(model.summary())
# NOTE: https://www.tensorflow.org/versions/r2.6/api_docs/python/tf/keras/Mod
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 60, 60, 32)        2432
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 56, 56, 32)        25632
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 28, 28, 32)        0
_________________________________________________________________
```

```
conv2d_2 (Conv2D)            (None, 26, 26, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 256)               524544
_________________________________________________________________
dense_1 (Dense)              (None, 200)               51400
=================================================================
Total params: 770,216
Trainable params: 770,216
Non-trainable params: 0
_________________________________________________________________
None
```

Running Infrence

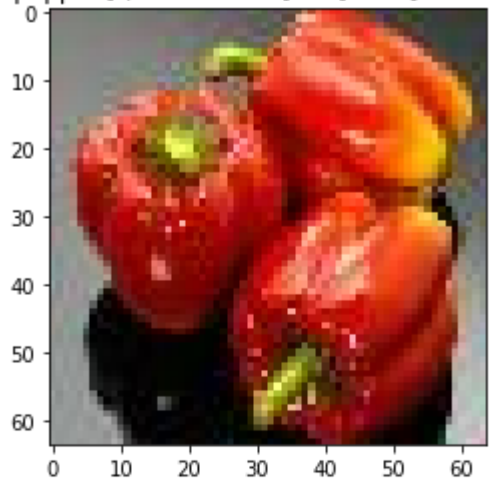
In [ ]:

```python
# Running infrence on our model
# We can run an infrence of our model by doing the following (we are doing ba
for example in ds_train.batch(1).take(1):
    img_info(example)

    # Make a prediction
    pred = model.predict(example["image"])
    print(f'Raw 200 Class Weighted Prediction:\n{pred}') # Uncomment to see t

    # What is out best guess?
    best_guess = tf.math.argmax(pred, axis=1).numpy() # Our output is 200 wei
    print(f'Best Guess [class index]: {class_names[best_guess[0]]} [{best_gue
    print(f'Best Guess Confidence (percent / 1.0): {pred[0][best_guess]}')

    # What are our top 15 guesses?
    top_15 = tf.math.top_k(pred, k=15)
    print(f'Top 15 Guesses (class index): {[f"{class_names[idx][0]} [{idx}]"
    print(f'Top 15 Guesses Confidence (percent / 1.0): {top_15.values}')
```

```
Label: [b'bell pepper'] (class index: [188] - id: [b'n07720875'])
Raw 200 Class Weighted Prediction:
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0.]]
Best Guess [class index]: ['pomegranate'] [170]
Best Guess Confidence (percent / 1.0): [1.]
```

```
Top 15 Guesses (class index): ['[ [170]', '[ [0]', '[ [1]', '[ [2]', '[ [3]',
'[ [4]', '[ [5]', '[ [6]', '[ [7]', '[ [8]', '[ [9]', '[ [10]', '[ [11]', '[
[12]', '[ [13]']
Top 15 Guesses Confidence (percent / 1.0): [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0.]]
```

[b'bell pepper'] (class index: [188] - id: [b'n07720875'])



In [ ]:

```
!who
```

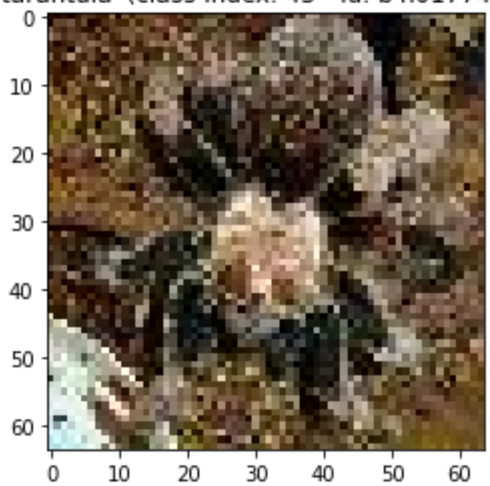
In [ ]:

```
# TODO: Run infrence for our previous 3 sample images

# TODO: Your Code Here
for img_data in sample_imgs: # Start a for-loop in order to iterate through o
    img_info(img_data)


    prediction = model.predict(np.expand_dims(img_data["image"], axis=0))

    best_guess = tf.math.argmax(prediction, axis=1).numpy() # Our output is 2
    print(f'Best Guess [class index]: {class_names[best_guess[0]]} [{best_gue
    print(f'Best Guess Confidence (percent / 1.0): {prediction[0][best_guess]

    # What are our top 15 guesses?
    top_15 = tf.math.top_k(prediction, k=15)
    print(f'Top 15 Guesses (class index): {[f"{class_names[idx][0]} [{idx}]"
    print(f'Top 15 Guesses Confidence (percent / 1.0): {top_15.values}')
```

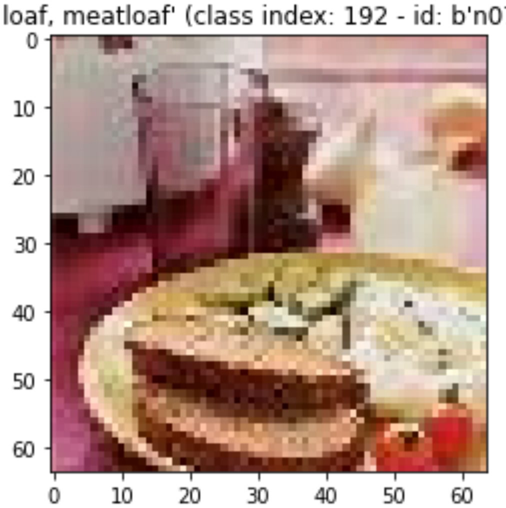
```
Label: b'tarantula' (class index: 43 - id: b'n01774750')
Best Guess [class index]: ['kimono'] [172]
Best Guess Confidence (percent / 1.0): [0.9999237]
Top 15 Guesses (class index): ['[ [172]', '[ [45]', '[ [173]', '[ [148]', '[
[10]', '[ [0]', '[ [1]', '[ [2]', '[ [3]', '[ [4]', '[ [5]', '[ [6]', '[
[7]', '[ [8]', '[ [9]']
Top 15 Guesses Confidence (percent / 1.0): [[9.9992371e-01 7.6236203e-05 1.19
01981e-19 8.2278266e-21 3.6131815e-36
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]]
Label: b'meat loaf, meatloaf' (class index: 192 - id: b'n07871810')
Best Guess [class index]: ['Christmas stocking'] [123]
Best Guess Confidence (percent / 1.0): [1.]
Top 15 Guesses (class index): ['[ [123]', '[ [63]', '[ [134]', '[ [61]', '[
```

```
[173]', '[ [45]', '[ [0]', '[ [1]', '[ [2]', '[ [3]', '[ [4]', '[ [5]', '[
[6]', '[ [7]', '[ [8]']
Top 15 Guesses Confidence (percent / 1.0): [[1.0000000e+00 1.5071672e-09 2.95
08073e-19 5.7862415e-20 7.9649574e-35
  1.9659285e-37 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]]
Label: b'sea slug, nudibranch' (class index: 165 - id: b'n01950731')
Best Guess [class index]: ['comic book'] [173]
Best Guess Confidence (percent / 1.0): [0.9515173]
Top 15 Guesses (class index): ['[ [173]', '[ [61]', '[ [63]', '[ [46]', '[ [1
34]', '[ [146]', '[ [143]', '[ [0]', '[ [1]', '[ [2]', '[ [3]', '[ [4]', '[
[5]', '[ [6]', '[ [7]']
Top 15 Guesses Confidence (percent / 1.0): [[9.5151728e-01 4.8462652e-02 2.00
52566e-05 1.9794566e-25 6.7907142e-27
  9.8127139e-33 4.6507435e-35 0.0000000e+00 0.0000000e+00 0.0000000e+00
```

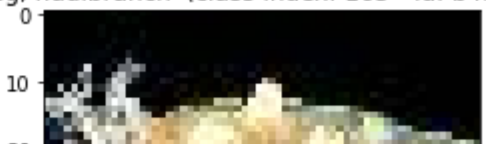
b'tarantula' (class index: 43 - id: b'n01774750')



b'meat loaf, meatloaf' (class index: 192 - id: b'n07871810')



b'sea slug, nudibranch' (class index: 165 - id: b'n01950731')



In [ ]:

```
# TODO: Calculate the Top-1, Top-5, and Top-10 Accuracy of the validation dat
total = acc_top1 = acc_top5 = acc_top10 = 0

# TODO: Your Code Here
for img_data in ds_val.batch(32): # Just creating another for loop in order t

    prediction = model.predict(img_data["image"].numpy())

    top_1 = tf.math.top_k(prediction, k=1).indices
    top_5 = tf.math.top_k(prediction, k=5).indices
    top_10 = tf.math.top_k(prediction, k=10).indices

    for i, img_label in enumerate(img_data['label'].numpy()):
        if img_label in top_1[i]:
            acc_top1 += 1
        if img_label in top_5[i]:
            acc_top5 += 1
        if img_label in top_10[i]:
            acc_top10 += 1

    total += 32


acc_top10 /= (total / 100) # Dividing by 1/100 multiplies :D
acc_top5 /= (total / 100)
acc_top1 /= (total / 100)

print("Accuracy (Top 10):", acc_top10, ",Accuracy (Top 5): ", acc_top5, ", Ac
```

```
Accuracy (Top 10): 15.225638977635784 ,Accuracy (Top 5):  12.649760383386582
, Accuracy (Top 1):  6.968849840255591
```

In [ ]:
```
# TODO: Print all of the possible classes of the dataset

train_classes = val_classes = 0

# TODO: Your Code Here
for image in ds_val:
    #class_name = image['metadata']['label_name'].numpy()
    val_classes += 1

# COMMENTED OUT FOR SUBMISSION

for image in ds_train:
#     class_name = image['metadata']['label_name'].numpy()
    train_classes += 1

print(class_names)
print("NUMBER OF CLASSES IN VAL DATASET: ", val_classes)
print("NUMBER OF CLASSES IN TRAIN DATASET", train_classes)
```

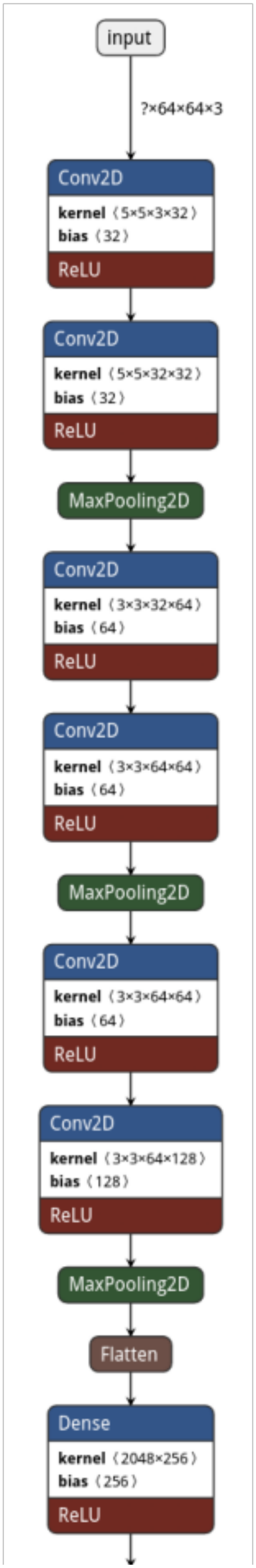
```
["['Egyptian cat']", "['reel']", "['volleyball']", "['rocking chair', 'rocker
']", "['lemon']", "['bullfrog', 'Rana catesbeiana']", "['basketball']", "['cl
iff', 'drop', 'drop-off']", "['espresso']", '[\'plunger\', "plumber\'s helpe
r"]', "['parking meter']", "['German shepherd', 'German shepherd dog', 'Germa
n police dog', 'alsatian']", "['dining table', 'board']", "['monarch', 'monar
ch butterfly', 'milkweed butterfly', 'Danaus plexippus']", "['brown bear', 'b
ruin', 'Ursus arctos']", "['school bus']", "['pizza', 'pizza pie']", "['guine
a pig', 'Cavia cobaya']", "['umbrella']", "['organ', 'pipe organ']", "['oboe
', 'hautboy', 'hautbois']", "['maypole']", "['goldfish', 'Carassius auratus
']", "['potpie']", "['hourglass']", "['seashore', 'coast', 'seacoast', 'sea-c
oast']", "['computer keyboard', 'keypad']", "['Arabian camel', 'dromedary', '
Camelus dromedarius']", "['ice cream', 'icecream']", "['nail']", "['space hea
ter']", "['cardigan']", "['baboon']", "['snail']", "['coral reef']", "['albat
ross', 'mollymawk']", '[\'spider web\', "spider\'s web"]', "['sea cucumber',
'holothurian']", "['backpack', 'back pack', 'knapsack', 'packsack', 'rucksack
', 'haversack']", "['Labrador retriever']", "['pretzel']", "['king penguin',
'Aptenodytes patagonica']", "['sulphur butterfly', 'sulfur butterfly']", "['t
arantula']", "['lesser panda', 'red panda', 'panda', 'bear cat', 'cat bear',
'Ailurus fulgens']", "['pop bottle', 'soda bottle']", "['banana']", "['sock
']", "['cockroach', 'roach']", "['projectile', 'missile']", "['beer bottle
']", "['mantis', 'mantid']", "['freight car']", "['guacamole']", "['remote co
ntrol', 'remote']", "['European fire salamander', 'Salamandra salamandra']",
"['lakeside', 'lakeshore']", "['chimpanzee', 'chimp', 'Pan troglodytes']",
"['pay-phone', 'pay-station']", "['fur coat']", "['alp']", "['lampshade', 'la
mp shade']", "['torch']", "['abacus']", "['moving van']", "['barrel', 'cask
']", "['tabby', 'tabby cat']", "['goose']", "['koala', 'koala bear', 'kangaro
o bear', 'native bear', 'Phascolarctos cinereus']", "['bullet train', 'bullet
']", "['CD player']", "['teapot']", "['birdhouse']", "['gazelle']", '[\'acade
mic gown\', \'academic robe\', "judge\'s robe"]', "['tractor']", "['ladybug',
'ladybeetle', 'lady beetle', 'ladybird', 'ladybird beetle']", "['miniskirt',
'mini']", "['golden retriever']", "['triumphal arch']", "['cannon']", "['neck
brace']", "['sombrero']", "['gasmask', 'respirator', 'gas helmet']", "['candl
e', 'taper', 'wax light']", "['desk']", "['frying pan', 'frypan', 'skillet
']", "['bee']", "['dam', 'dike', 'dyke']", "['spiny lobster', 'langouste', 'r
ock lobster', 'crawfish', 'crayfish', 'sea crawfish']", "['police van', 'poli
ce wagon', 'paddy wagon', 'patrol wagon', 'wagon', 'black Maria']", "['iPod
']", "['punching bag', 'punch bag', 'punching ball', 'punchball']", "['beacon
', 'lighthouse', 'beacon light', 'pharos']", "['jellyfish']", "['wok']", '["p
otter\'s wheel"]', "['sandal']", "['pill bottle']", "['butcher shop', 'meat m
```

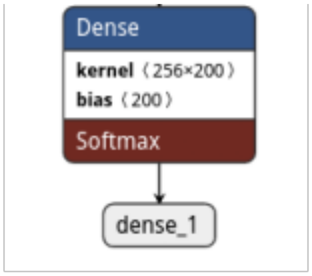
```
arket']", "['slug']", "['hog', 'pig', 'grunter', 'squealer', 'Sus scrofa']",
"['cougar', 'puma', 'catamount', 'mountain lion', 'painter', 'panther', 'Feli
s concolor']", "['crane']", "['vestment']", '[\'dragonfly\', \'darning needl
e\', "devil\'s darning needle", \'sewing needle\', \'snake feeder\', \'snake
doctor\', \'mosquito hawk\', \'skeeter hawk\']', "['cash machine', 'cash disp
enser', 'automated teller machine', 'automatic teller machine', 'automated te
ller', 'automatic teller', 'ATM']", "['mushroom']", "['jinrikisha', 'ricksha
', 'rickshaw']", "['water tower']", "['chest']", "['snorkel']", "['sunglasses
', 'dark glasses', 'shades']", "['fly']", "['limousine', 'limo']", "['black s
tork', 'Ciconia nigra']", "['dugong', 'Dugong dugon']", "['sports car', 'spor
t car']", "['water jug']", "['suspension bridge']", "['ox']", "['ice lolly',
'lolly', 'lollipop', 'popsicle']", "['turnstile']", "['Christmas stocking']",
"['broom']", "['scorpion']", "['wooden spoon']", "['picket fence', 'paling
']", "['rugby ball']", "['sewing machine']", "['steel arch bridge']", "['Pers
ian cat']", "['refrigerator', 'icebox']", "['barn']", "['apron']", "['Yorkshi
re terrier']", "['swimming trunks', 'bathing trunks']", "['stopwatch', 'stop
watch']", "['lawn mower', 'mower']", "['thatch', 'thatched roof']", "['founta
in']", "['black widow', 'Latrodectus mactans']", "['bikini', 'two-piece']",
"['plate']", "['teddy', 'teddy bear']", "['barbershop']", "['confectionery',
'confectionary', 'candy store']", "['beach wagon', 'station wagon', 'wagon',
'estate car', 'beach waggon', 'station waggon', 'waggon']", "['scoreboard']",
"['orange']", "['flagpole', 'flagstaff']", "['American lobster', 'Northern lo
bster', 'Maine lobster', 'Homarus americanus']", "['trolleybus', 'trolley coa
ch', 'trackless trolley']", "['drumstick']", "['dumbbell']", "['brass', 'memo
rial tablet', 'plaque']", "['bow tie', 'bow-tie', 'bowtie']", "['convertible
']", "['bighorn', 'bighorn sheep', 'cimarron', 'Rocky Mountain bighorn', 'Roc
ky Mountain sheep', 'Ovis canadensis']", "['orangutan', 'orang', 'orangutang
', 'Pongo pygmaeus']", "['American alligator', 'Alligator mississipiensis']",
"['centipede']", "['syringe']", "['go-kart']", "['brain coral']", "['sea slug
', 'nudibranch']", "['cliff dwelling']", "['mashed potato']", "['viaduct']",
"['military uniform']", "['pomegranate']", "['chain']", "['kimono']", "['comi
c book']", "['trilobite']", "['bison']", "['pole']", "['boa constrictor', 'Co
nstrictor constrictor']", "['poncho']", "['bathtub', 'bathing tub', 'bath', '
tub']", "['grasshopper', 'hopper']", "['walking stick', 'walkingstick', 'stic
k insect']", "['Chihuahua']", "['tailed frog', 'bell toad', 'ribbed toad', 't
ailed toad', 'Ascaphus trui']", "['lion', 'king of beasts', 'Panthera leo']",
"['altar']", "['obelisk']", "['beaker']", "['bell pepper']", "['bannister', '
banister', 'balustrade', 'balusters', 'handrail']", "['bucket', 'pail']", "['
magnetic compass']", "['meat loaf', 'meatloaf']", "['gondola']", "['standard
poodle']", "['acorn']", "['lifeboat']", "['binoculars', 'field glasses', 'ope
ra glasses']", "['cauliflower']", "['African elephant', 'Loxodonta africana
']"]
NUMBER OF CLASSES IN VAL DATASET:  10000
NUMBER OF CLASSES IN TRAIN DATASET 100000
```

Model Exploration

In [ ]:

```python
# TODO: Visualize the model in Netron and include an image here.
tf.keras.utils.plot_model(model, "model.png", show_shapes=True, show_dtype=Tr

#My code for printing out the image created in Netron, using pyplot
plt.figure(figsize=(20, 20))
netron_image = plt.imread("netron_model.png") # Grabs saved image
plt.imshow(netron_image) # Displays the image
plt.axis('off') # Disables axis formatting in plot
plt.show() # Displays image
```

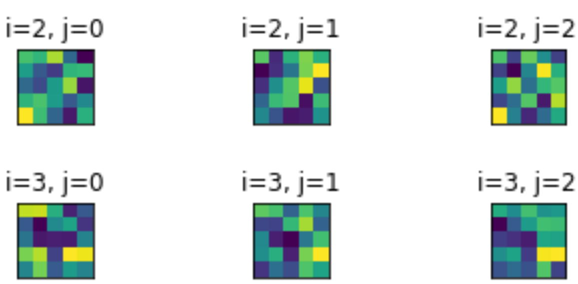

input
?×64×64×3
Conv2D
kernel ⟨5×5×3×32⟩
bias ⟨32⟩
ReLU
Conv2D
kernel ⟨5×5×32×32⟩
bias ⟨32⟩
ReLU
MaxPooling2D
Conv2D
kernel ⟨3×3×32×64⟩
bias ⟨64⟩
ReLU
Conv2D
kernel ⟨3×3×64×64⟩
bias ⟨64⟩
ReLU
MaxPooling2D
Conv2D
kernel ⟨3×3×64×64⟩
bias ⟨64⟩
ReLU
Conv2D
kernel ⟨3×3×64×128⟩
bias ⟨128⟩
ReLU
MaxPooling2D
Flatten
Dense
kernel ⟨2048×256⟩
bias ⟨256⟩
ReLU



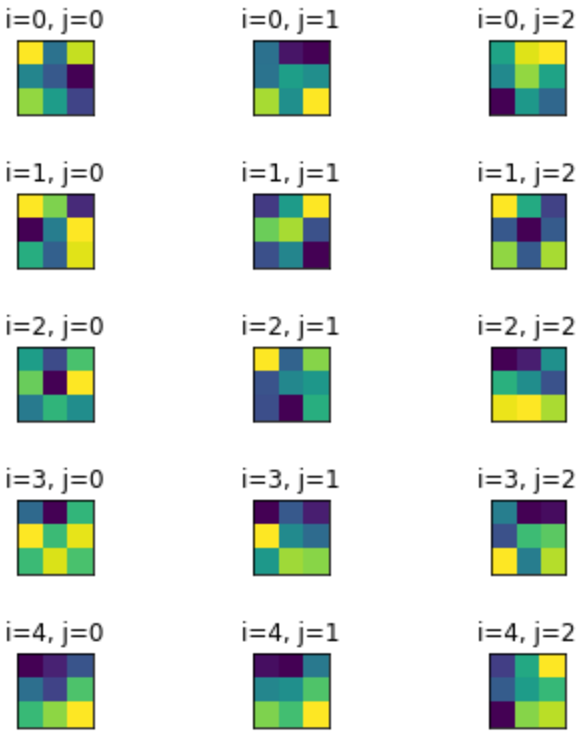
In [ ]:

```
# We can view the layer weights as well. Here we are pretending they are imag
# TODO: Visualize the 2 convolutional layers filter sets (weights) (one at th

# TODO: Your Code Here

# Begin with our beginning input layer
# beginning_layer = model.layers[3]
# filter_weight, filter_biases = beginning_layer.get_weights()

# print(f'Beginning Layer: {beginning_layer.name}') # Prove name of layer
# print(f'Data Type: {filter_weight.dtype}') # Print dtype
# print(f'Dimensions: {filter_weight.ndim} total {filter_weight.shape}') # Pr
# print(f'Required memory: {filter_weight.nbytes}B ({filter_weight.nbytes/100

# filter_min, filter_max = filter_weight.min(), filter_weight.max()
# filter_weight = (filter_weight - filter_min) / (filter_max - filter_min)
# n_filters, i = 14, 1

# for foo in range(1, n_filters):
#     f = filter_weight[:, :, :, foo]
#     for j in range(3):
#         ax = plt.subplot(n_filters, 5, foo)
#         ax.set_xticks([])
#         ax.set_yticks([])

#         plt.imshow(f[:, :, j])
#         foo += 1
# plt.figure(figsize=(160, 160), dpi=10).show()

# # End with our later layer
# later_layer = model.layers[7]
# filter_weight, filter_biases = later_layer.get_weights()

# print(f'Ending Layer: {later_layer.name}') # Prove name of layer
# print(f'Data Type: {filter_weight.dtype}') # Print dtype
# print(f'Dimensions: {filter_weight.ndim} total {filter_weight.shape}') # Pr
# print(f'Required memory: {filter_weight.nbytes}B ({filter_weight.nbytes/100

# filter_min, filter_max = filter_weight.min(), filter_weight.max()
# later_layer = (filter_weight - filter_min) / (filter_max - filter_min)
# n_filters, i = 22, 1

# for foo in range(1, n_filters):
#     f = filter_weight [:, :, :, foo]
#     for j in range(5):
#         ax = plt.subplot(n_filters, 5, foo)
#         ax.set_xticks([])
#         ax.set_yticks([])
```

```
#         plt.imshow(f[:, :, j])
#         foo += 1
# plt.figure(figsize=(160, 160), dpi=10).show()

filters, biases = model.layers[0].get_weights()
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)

n_filters, ix = 5, 1
for i in range(n_filters):
    f = filters[:, :, :, i]
    for j in range(3):
        ax = plt.subplot(n_filters, 3, ix)
        ax.set_xticks([])
        ax.set_yticks([])
        plt.imshow(f[:, :, j])
        plt.title(f'i={i}, j={j}')
        ix += 1
    plt.show()

print(f'Beginning Layer: {model.layers[0].name}') # Prove name of layer
print(f'Data Type: {filters.dtype}') # Print dtype
print(f'Dimensions: {filters.ndim} total {filters.shape}') # Print #dims, + s
print(f'Required memory: {filters.nbytes}B ({filters.nbytes/1000}KB)') # Prin

filters, biases = model.layers[7].get_weights()
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)

n_filters, ix = 5, 1
for i in range(n_filters):
    f = filters[:, :, :, i]
    for j in range(3):
        ax = plt.subplot(n_filters, 3, ix)
        ax.set_xticks([])
        ax.set_yticks([])
        plt.imshow(f[:, :, j])
        plt.title(f'i={i}, j={j}')
        ix += 1
    plt.show()

print(f'Beginning Layer: {model.layers[7].name}') # Prove name of layer
print(f'Data Type: {filters.dtype}') # Print dtype
print(f'Dimensions: {filters.ndim} total {filters.shape}') # Print #dims, + s
print(f'Required memory: {filters.nbytes}B ({filters.nbytes/1000}KB)') # Prin
```



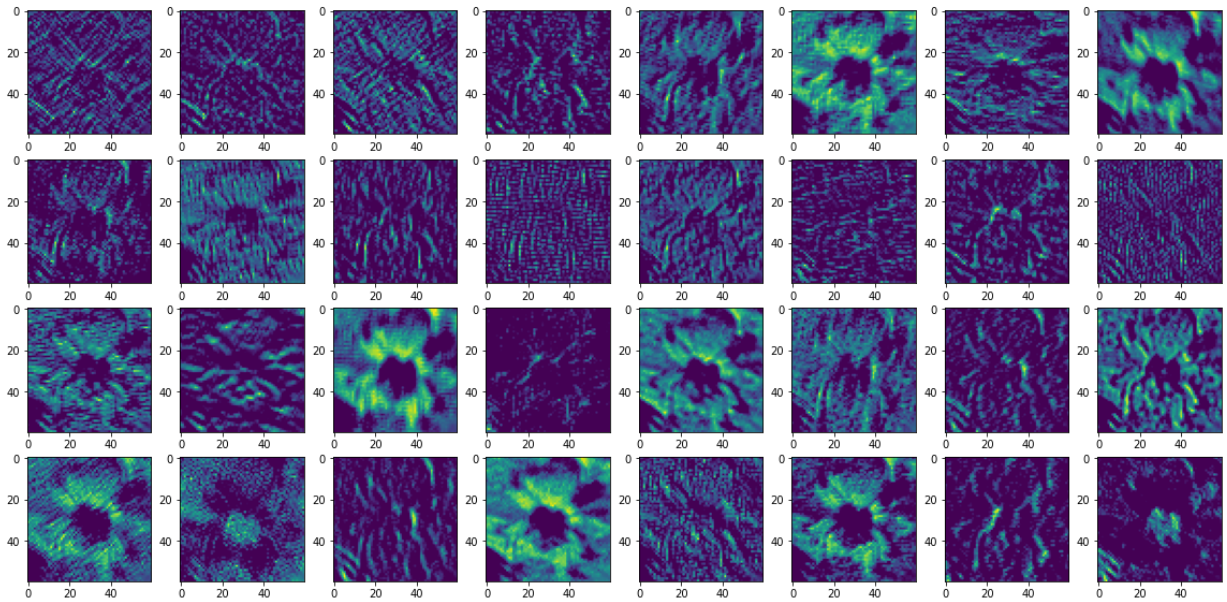

```
Beginning Layer: conv2d
Data Type: float32
Dimensions: 4 total (5, 5, 3, 32)
Required memory: 9600B (9.6KB)
```



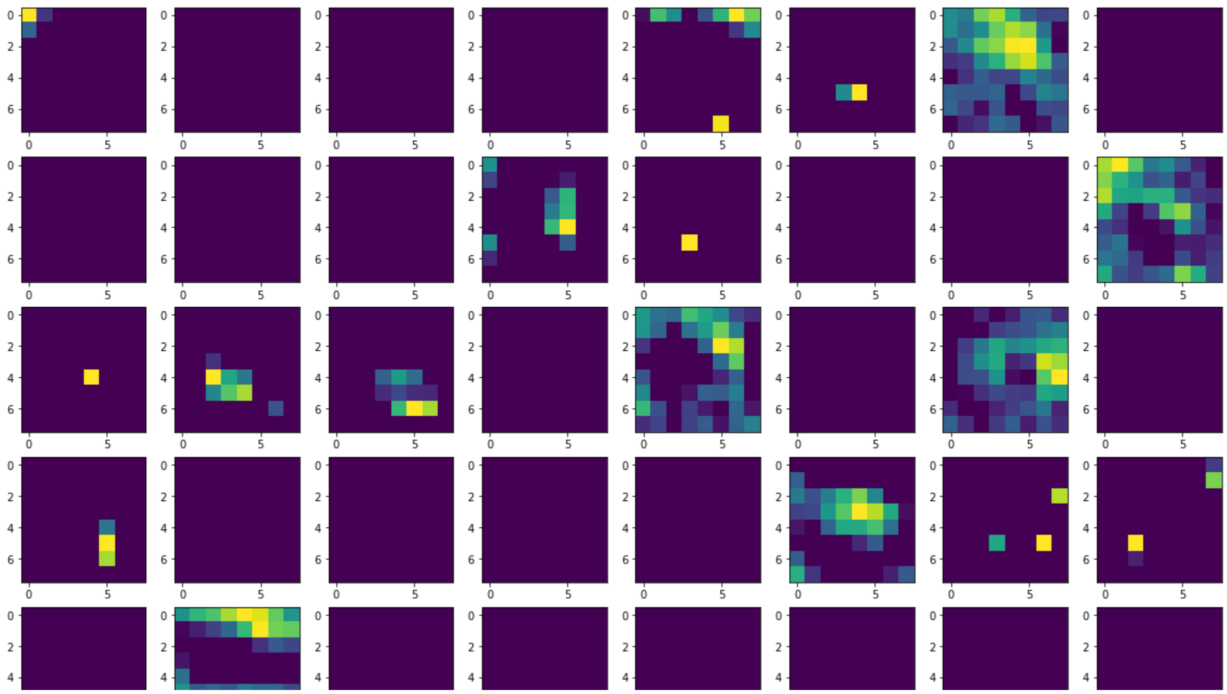

```
Beginning Layer: conv2d_5
```

In [ ]:
```
# We can again view the layer outputs as well. Here we are pretending they ar
# TODO: Visualize the 2 convolutional layers outputs (intermediate feature ma


# TODO: Your Code Here
# test_model = tf.keras.applications.vgg16.VGG16()

# for i in range(len(model.layers)):
#     layer = model.layers[i]
#     if 'conv' not in layer.name:
#         continue
#     print(i , layer.name , layer.output.shape)

# Run inference to get feature maps on earlier layer
sub_model = Model(inputs=model.inputs, outputs=model.layers[0].output)

#print(f"Printing feature maps for layer {model.layers[0].name}...")

image = sample_imgs[0]['image']
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)
image = tf.keras.applications.vgg16.preprocess_input(image)

feature_maps = sub_model.predict(image)
idx = 1
fig = plt.figure(figsize=(20, 20))
for i in range (1, feature_maps.shape[3]+1):

    if idx == 65:
        break

    plt.subplot(8, 8, i)
    plt.imshow(feature_maps[0, :, :, i-1])
    idx += 1


plt.show()

print(f'Beginning Layer: {model.layers[0].name}') # Prove name of layer
print(f'Data Type: {feature_maps.dtype}') # Print dtype
print(f'Dimensions: {feature_maps.ndim} total {feature_maps.shape}') # Print
print(f'Required memory: {feature_maps.nbytes}B ({feature_maps.nbytes/1000}KB

# Run infrence to get feature maps on end conv layer

sub_model = Model(inputs=model.input, outputs=model.layers[7].output)

#print(f"Printing feature maps for layer {model.layers[7].name}...")

image = sample_imgs[0]['image']
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)
image = tf.keras.applications.vgg16.preprocess_input(image)

feature_maps = sub_model.predict(image)

idx = 1
fig = plt.figure(figsize=(20, 20))
```

```
for i in range (1, feature_maps.shape[3]+1):

    if idx == 65:
        break

    plt.subplot(8, 8, i)
    plt.imshow(feature_maps[0, :, :, i-1])
    idx += 1

plt.show()

print(f'End Layer: {model.layers[7].name}') # Prove name of layer
print(f'Data Type: {feature_maps.dtype}') # Print dtype
print(f'Dimensions: {feature_maps.ndim} total {feature_maps.shape}') # Print
print(f'Required memory: {feature_maps.nbytes}B ({feature_maps.nbytes/1000}KB
```



```
Beginning Layer: conv2d
Data Type: float32
Dimensions: 4 total (1, 60, 60, 32)
Required memory: 460800B (460.8KB)
```



In [ ]:

```
# TODO: Export the filters/weights se we can use them later
# Make a directory for our image data
model_dir = os.path.abspath('model_data')
pathlib.Path(model_dir).mkdir(exist_ok=True)


# Export each image
conv_index = dense_index = 1 # layer index starts from one
for layer_idx, layer in enumerate(model.layers):
    if re.match("conv", layer.name):
        weight_file_name = os.path.join(model_dir, f'conv{conv_index}_weights
        bias_file_name = os.path.join(model_dir, f'conv{conv_index}_bias.bin'
        conv_index += 1
    elif re.match("dense", layer.name):
        weight_file_name = os.path.join(model_dir, f'dense{dense_index}_weigh
        bias_file_name = os.path.join(model_dir, f'dense{dense_index}_bias.bi
        dense_index += 1
    else:
        continue
    # INPUT CODE BELOW
    weights, biases = layer.get_weights()
    weight_data, bias_data = weights.flatten(), biases.flatten()
    weight_file = open(weight_file_name, 'wb').write(weight_data.tobytes())
    bias_file = open(bias_file_name, 'wb').write(bias_data.tobytes())


print(f"All the convolution and dense (fully connected) weights and biases su
```

```
All the convolution and dense (fully connected) weights and biases successful
ly exported to input folders in /home/tjfriedl/Desktop/cpre_487/lab1/model_da
ta directory
```

In [ ]:

```
# TODO: Export the intermediate layer outputs for each of the input for all o
img_dir = os.path.abspath('img_data')
pathlib.Path(img_dir).mkdir(exist_ok=True)

for img_idx, img in enumerate(sample_imgs):
    file_dir = os.path.join(img_dir, f'test_input_{img_idx}')
    pathlib.Path(file_dir).mkdir(exist_ok=True)

    for layer_idx, layer in enumerate(model.layers):
        aux_model = tf.keras.Model(inputs=model.inputs, outputs=[layer.output

        # Store the intermediate output

        # TODO: Your Code Here
        aux_image = tf.keras.preprocessing.image.img_to_array(img['image'])
        aux_image = np.expand_dims(aux_image, axis=0)
        aux_image = tf.keras.applications.vgg16.preprocess_input(aux_image)

        feature_maps = aux_model.predict(aux_image)

        fmap_data = feature_maps.flatten()
        fmap_file_name = os.path.join(file_dir, f'layer_output_{layer_idx}.bi
        fmap_file = open(fmap_file_name, 'wb').write(fmap_data.tobytes())


print(f"All the corresponding intermediate layer outputs successfully exporte
```

```
All the corresponding intermediate layer outputs successfully exported to eac
h input folder in the /home/tjfriedl/Desktop/cpre_487/lab1/img_data directory
```

Tensorboard

In [ ]:

```
# Setup for profiling
tf.profiler.experimental.ProfilerOptions(
    host_tracer_level=1, python_tracer_level=0, device_tracer_level=1
)

log_dir = os.path.abspath(os.path.join('log_data', datetime.datetime.now().st
pathlib.Path(log_dir).mkdir(exist_ok=True, parents=True)
```

In [ ]:

```
# TODO: Sample Profiling - Inference for a single image:

# Perform the inference profiling:

# THE ABOVE EXAMPLE IS DONE FOR ONLY A SINGLE TEST IMAGE FROM ds_train.batch(

for example in ds_train.batch(1).take(1):
    # Starts Profile logging
    tf.profiler.experimental.start(os.path.join(log_dir, f'single-{datetime.d

    # Actual inference
    # TODO: Your Code Here
    model.predict(example['image'])


    # Stops Profile logging
    tf.profiler.experimental.stop()

# THE BELOW EXAMPLE IS DONE FOR OUR THREE SAMPLE IMAGES COMING FROM sample_im
# tf.profiler.experimental.start(os.path.join(log_dir, f'single-{datetime.dat
# for image in sample_imgs:
#     # Starts Profile logging
#     #tf.profiler.experimental.start(os.path.join(log_dir, f'single-{datetim

#     #Actual inference
#     # TODO: Your Code Here
#     img = tf.keras.preprocessing.image.img_to_array(image['image'])
#     img = np.expand_dims(img, axis=0)
#     img = tf.keras.applications.vgg16.preprocess_input(img)

#     model.predict(img)

#     #  Stops Profile logging
# tf.profiler.experimental.stop()


# Load the TensorBoard notebook extension.
%load_ext tensorboard

# Launch TensorBoard and navigate to the Profile tab to view performance prof
# *** Please note just execute this command ones in a session and
# then logs for subsequent runs would be auto detected in tensorboard- url: h
%tensorboard --logdir=log_dir

# You could view the tensorboard in the browser url: http://localhost:6006/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 226879), started 5 days, 20:47:26 ago.
(Use '!kill 226879' to kill it.)
```

In [ ]:

```
# TODO: Sample Profiling - Online Inference:

# Vary this from 10, 100, 1000 to simulate multiple online inference
loop_index = [10, 100, 1000]

for idx in loop_index:
    # Starts Profile logging
    # tf.profiler.experimental.stop()
    tf.profiler.experimental.start(os.path.join(log_dir, f'online-infrence-{i

    # Actual online inference
    # TODO: Your Code Here
    for index, example in enumerate(ds_val.take(idx)):
        sample = tf.keras.preprocessing.image.img_to_array(example['image'])
        sample = np.expand_dims(sample, axis=0)
        sample = tf.keras.applications.vgg16.preprocess_input(sample)

        model.predict(sample)
    print(f'BATCH CALCULATION {idx} COMPLETE')


    # Stops Profile logging
    tf.profiler.experimental.stop()

# Load the TensorBoard notebook extension.
%load_ext tensorboard

# Launch TensorBoard and navigate to the Profile tab to view performance prof
# *** Please note just execute this command ones in a session and
# then logs for subsequent runs would be auto detected in tensorboard- url: h
%tensorboard --logdir=log_dir

# You could view the tensorboard in the browser url: http://localhost:6006/ a
```

```
BATCH CALCULATION 10 COMPLETE
BATCH CALCULATION 100 COMPLETE
BATCH CALCULATION 1000 COMPLETE
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 226879), started 5 days, 20:47:58 ago.
(Use '!kill 226879' to kill it.)
```

In [ ]:

```
# TODO: Sample Profiling - Batch Inference:

# We would only perform batch inference for a subset of validation set i.e. 1
# using different batch sizes of 20, 40, 100, 200

# Decides the size of the batch. Try: 20, 40, 100, 200
batch_size = [20, 40, 100, 200]

for batch in batch_size:
    # Starts Profile logging

    tf.profiler.experimental.start(os.path.join(log_dir, f'batch-{batch}-{dat

    # Actual Batch inference
    # TODO: Your Code Here
    for example in enumerate(ds_val.batch(batch).take(1000)):
        model.predict(sample)
    print(f'BATCH SIZE {batch} COMPLETE')

    # Stops Profile logging
    tf.profiler.experimental.stop()

# Load the TensorBoard notebook extension.
%load_ext tensorboard

# Launch TensorBoard and navigate to the Profile tab to view performance prof
# *** Please note just execute this command ones in a session and
# then logs for subsequent runs would be auto detected in tensorboard- url: h
%tensorboard --logdir=log_dir

# You could view the tensorboard in the browser url: http://localhost:6006/ a
```

```
BATCH SIZE 20 COMPLETE
BATCH SIZE 40 COMPLETE
BATCH SIZE 100 COMPLETE
BATCH SIZE 200 COMPLETE
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 226879), started 5 days, 20:48:29 ago.
(Use '!kill 226879' to kill it.)
```

Training

In [ ]:

```
# Setup for model training
from tensorflow.keras import Model, datasets
from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense, Flatten, Conv2D, AveragePooling2D,

train_dir = os.path.abspath(os.path.join('train_data', datetime.datetime.now(
pathlib.Path(train_dir).mkdir(exist_ok=True, parents=True)

# Using early stopping to monitor validation accuracy
callbacks = [
    tf.keras.callbacks.EarlyStopping(
        # Stop training when `val_loss` is no longer improving
        monitor="val_loss",
        # "no longer improving" being defined as "no better than 1e-2 less"
        min_delta=1e-4,
        # "no longer improving" being further defined as "for at least 2 epoc
        patience=2,
        verbose=1,
    ),
    tf.keras.callbacks.TensorBoard(log_dir=train_dir, histogram_freq=1)
]
```

In [ ]:

```
# Basic CNN model
train_model = Sequential()

# conv1
train_model.add(Conv2D(32, (5, 5), input_shape=(64, 64, 3), activation='relu'
train_model.add(Conv2D(32, (5,5),activation='relu'))
train_model.add(MaxPooling2D(pool_size=(2, 2)))
train_model.add(Conv2D(64, (3,3), activation='relu'))
train_model.add(Conv2D(64, (3,3), activation='relu'))
train_model.add(MaxPooling2D(pool_size=(2, 2)))
train_model.add(Conv2D(64, (3,3), activation='relu'))
train_model.add(Conv2D(128, (3,3), activation='relu'))
train_model.add(MaxPooling2D(pool_size=(2, 2)))
train_model.add(Flatten())

# fc1
train_model.add(Dense(256, activation='relu'))

# fc2
train_model.add(Dense(200, activation='softmax'))

train_model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optim
train_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 60, 60, 32)        2432
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 56, 56, 32)        25632
_________________________________________________________________
```

```
max_pooling2d_6 (MaxPooling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 26, 26, 64)        18496
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 24, 24, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               524544
_________________________________________________________________
dense_5 (Dense)              (None, 200)               51400
=================================================================
Total params: 770,216
Trainable params: 770,216
Non-trainable params: 0
```

In [ ]:

```
# TODO: Attempt to train your own model with different batch sizes


def normalize_img(image, label):
    return tf.cast(image, tf.float32) / 255., label

def to_categorical(image, label):
    label = tf.one_hot(tf.cast(label, tf.int32), 200)
    return tf.cast(image, tf.float32), tf.cast(label, tf.int64)

ds_re = tiny_imagenet_builder.as_dataset(as_supervised=True)
ds_retrain, ds_reval = ds_re["train"], ds_re["validation"]

ds_retrain = ds_retrain.cache().shuffle(1024)
ds_reval = ds_reval.cache().shuffle(1024)

ds_retrain = ds_retrain.map(normalize_img, num_parallel_calls=tf.data.AUTOTUN
ds_reval = ds_reval.map(normalize_img, num_parallel_calls=tf.data.AUTOTUNE)

ds_retrain = ds_retrain.map(to_categorical, num_parallel_calls=tf.data.AUTOTU
ds_reval = ds_reval.map(to_categorical, num_parallel_calls=tf.data.AUTOTUNE)

epoch_size = 20

weights = train_model.get_weights()


for batch_size in [32, 64, 128]:
    # Setup our batched datasets
    # TODO: Your Code Here

    print(f"BATCH SIZE {batch_size}:")
    retrain = ds_retrain.batch(batch_size, num_parallel_calls=tf.data.AUTOTUN
    reval = ds_reval.batch(batch_size, num_parallel_calls=tf.data.AUTOTUNE)

    # Run training
    # TODO: Your Code Here
    train_model.set_weights(weights=weights)

    train_model.fit(x=retrain, validation_data=reval, epochs=epoch_size, call

    # Save the cnn model
    train_model.save(os.path.join(log_dir, f'CNN_TinyImageNet_train_batch{bat

    # TODO: Get the top-1 and top-5 of your newly trained model
    # TODO: Your Code Here

    total = acc_top1 = acc_top5 = 0

    # TODO: Your Code Here
    for image in ds_val.batch(batch_size):
        prediction = train_model.predict(image['image'].numpy())

        top_1 = tf.math.top_k(prediction, k=1).indices
        top_5 = tf.math.top_k(prediction, k=5).indices

        for index, labels in enumerate(image['label'].numpy()):
            if labels in top_1[index]:
```

```
                acc_top1 += 1
            if labels in top_5[index]:
                acc_top5 += 1

        total += batch_size

    print(f"Top 1: {(acc_top1 / total)* 100:.2f}%, Top 5: {(acc_top5 / total)
```

```
BATCH SIZE 32:
Epoch 1/20
3125/3125 [==============================] - 18s 6ms/step - loss: 4.9877 - ac
curacy: 0.0294 - val_loss: 4.6644 - val_accuracy: 0.0615
Epoch 2/20
3125/3125 [==============================] - 17s 5ms/step - loss: 4.4464 - ac
curacy: 0.0891 - val_loss: 4.3280 - val_accuracy: 0.1081
Epoch 3/20
3125/3125 [==============================] - 16s 5ms/step - loss: 4.1393 - ac
curacy: 0.1311 - val_loss: 4.0306 - val_accuracy: 0.1478
Epoch 4/20
3125/3125 [==============================] - 17s 5ms/step - loss: 3.8915 - ac
curacy: 0.1685 - val_loss: 3.8682 - val_accuracy: 0.1715
Epoch 5/20
3125/3125 [==============================] - 16s 5ms/step - loss: 3.6877 - ac
curacy: 0.1999 - val_loss: 3.7318 - val_accuracy: 0.1915
Epoch 6/20
3125/3125 [==============================] - 16s 5ms/step - loss: 3.5176 - ac
curacy: 0.2264 - val_loss: 3.6333 - val_accuracy: 0.2064
Epoch 7/20
3125/3125 [==============================] - 16s 5ms/step - loss: 3.3771 - ac
curacy: 0.2500 - val_loss: 3.5579 - val_accuracy: 0.2213
Epoch 8/20
3125/3125 [==============================] - 16s 5ms/step - loss: 3.2516 - ac
curacy: 0.2700 - val_loss: 3.5205 - val_accuracy: 0.2274
Epoch 9/20
3125/3125 [==============================] - 16s 5ms/step - loss: 3.1331 - ac
curacy: 0.2905 - val_loss: 3.4480 - val_accuracy: 0.2410
Epoch 10/20
3125/3125 [==============================] - 16s 5ms/step - loss: 3.0244 - ac
curacy: 0.3100 - val_loss: 3.4701 - val_accuracy: 0.2443
Epoch 11/20
3125/3125 [==============================] - 16s 5ms/step - loss: 2.9181 - ac
curacy: 0.3292 - val_loss: 3.4718 - val_accuracy: 0.2477
Epoch 00011: early stopping
Top 1: 12.57%, Top 5: 18.51%
BATCH SIZE 64:
Epoch 1/20
1563/1563 [==============================] - 12s 8ms/step - loss: 5.2673 - ac
curacy: 0.0068 - val_loss: 5.1256 - val_accuracy: 0.0142
Epoch 2/20
1563/1563 [==============================] - 12s 7ms/step - loss: 5.0439 - ac
curacy: 0.0237 - val_loss: 4.9358 - val_accuracy: 0.0363
Epoch 3/20
1563/1563 [==============================] - 11s 7ms/step - loss: 4.8173 - ac
curacy: 0.0504 - val_loss: 4.7299 - val_accuracy: 0.0615
Epoch 4/20
1563/1563 [==============================] - 12s 7ms/step - loss: 4.5471 - ac
curacy: 0.0832 - val_loss: 4.4268 - val_accuracy: 0.0997
Epoch 5/20
1563/1563 [==============================] - 11s 7ms/step - loss: 4.2747 - ac
```

```
curacy: 0.1167 - val_loss: 4.1854 - val_accuracy: 0.1235
Epoch 6/20
1563/1563 [==============================] - 12s 7ms/step - loss: 4.0641 - ac
curacy: 0.1456 - val_loss: 4.0385 - val_accuracy: 0.1456
Epoch 7/20
1563/1563 [==============================] - 12s 7ms/step - loss: 3.8868 - ac
curacy: 0.1724 - val_loss: 3.9180 - val_accuracy: 0.1645
Epoch 8/20
1563/1563 [==============================] - 12s 7ms/step - loss: 3.7308 - ac
curacy: 0.1966 - val_loss: 3.8403 - val_accuracy: 0.1770
Epoch 9/20
1563/1563 [==============================] - 12s 8ms/step - loss: 3.5929 - ac
curacy: 0.2165 - val_loss: 3.6954 - val_accuracy: 0.2025
Epoch 10/20
1563/1563 [==============================] - 12s 7ms/step - loss: 3.4696 - ac
curacy: 0.2393 - val_loss: 3.6662 - val_accuracy: 0.2073
Epoch 11/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.3604 - ac
curacy: 0.2579 - val_loss: 3.6044 - val_accuracy: 0.2197
Epoch 12/20
1563/1563 [==============================] - 12s 7ms/step - loss: 3.2565 - ac
curacy: 0.2756 - val_loss: 3.5643 - val_accuracy: 0.2205
Epoch 13/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.1687 - ac
curacy: 0.2913 - val_loss: 3.5538 - val_accuracy: 0.2293
Epoch 14/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.0743 - ac
curacy: 0.3067 - val_loss: 3.5116 - val_accuracy: 0.2359
Epoch 15/20
1563/1563 [==============================] - 12s 7ms/step - loss: 2.9883 - ac
curacy: 0.3237 - val_loss: 3.5024 - val_accuracy: 0.2398
Epoch 16/20
1563/1563 [==============================] - 12s 7ms/step - loss: 2.8992 - ac
curacy: 0.3388 - val_loss: 3.5197 - val_accuracy: 0.2418
Epoch 17/20
1563/1563 [==============================] - 12s 7ms/step - loss: 2.8182 - ac
curacy: 0.3521 - val_loss: 3.5207 - val_accuracy: 0.2461
Epoch 00017: early stopping
Top 1: 15.63%, Top 5: 22.04%
BATCH SIZE 128:
Epoch 1/20
782/782 [==============================] - 10s 12ms/step - loss: 5.2967 - acc
uracy: 0.0053 - val_loss: 5.2869 - val_accuracy: 0.0051
Epoch 2/20
782/782 [==============================] - 9s 12ms/step - loss: 5.1960 - accu
racy: 0.0101 - val_loss: 5.1091 - val_accuracy: 0.0124
Epoch 3/20
782/782 [==============================] - 9s 12ms/step - loss: 5.0709 - accu
racy: 0.0205 - val_loss: 5.0285 - val_accuracy: 0.0228
Epoch 4/20
782/782 [==============================] - 9s 12ms/step - loss: 4.9648 - accu
racy: 0.0328 - val_loss: 4.8737 - val_accuracy: 0.0418
Epoch 5/20
782/782 [==============================] - 9s 12ms/step - loss: 4.7869 - accu
racy: 0.0546 - val_loss: 4.7144 - val_accuracy: 0.0648
Epoch 6/20
782/782 [==============================] - 9s 12ms/step - loss: 4.5699 - accu
racy: 0.0800 - val_loss: 4.4717 - val_accuracy: 0.0930
Epoch 7/20
782/782 [==============================] - 9s 12ms/step - loss: 4.3744 - accu
```

```
racy: 0.1034 - val_loss: 4.3184 - val_accuracy: 0.1139
Epoch 8/20
782/782 [==============================] - 9s 12ms/step - loss: 4.2085 - accu
racy: 0.1270 - val_loss: 4.1920 - val_accuracy: 0.1268
Epoch 9/20
782/782 [==============================] - 9s 12ms/step - loss: 4.0655 - accu
racy: 0.1463 - val_loss: 4.1014 - val_accuracy: 0.1444
Epoch 10/20
782/782 [==============================] - 9s 12ms/step - loss: 3.9341 - accu
racy: 0.1668 - val_loss: 4.0131 - val_accuracy: 0.1532
Epoch 11/20
782/782 [==============================] - 9s 12ms/step - loss: 3.8194 - accu
racy: 0.1855 - val_loss: 3.9124 - val_accuracy: 0.1684
Epoch 12/20
782/782 [==============================] - 9s 12ms/step - loss: 3.7083 - accu
racy: 0.2013 - val_loss: 3.8172 - val_accuracy: 0.1824
Epoch 13/20
782/782 [==============================] - 9s 12ms/step - loss: 3.6082 - accu
racy: 0.2195 - val_loss: 3.7911 - val_accuracy: 0.1848
Epoch 14/20
782/782 [==============================] - 9s 12ms/step - loss: 3.5176 - accu
racy: 0.2328 - val_loss: 3.7242 - val_accuracy: 0.1980
Epoch 15/20
782/782 [==============================] - 9s 12ms/step - loss: 3.4331 - accu
racy: 0.2462 - val_loss: 3.6923 - val_accuracy: 0.2025
Epoch 16/20
782/782 [==============================] - 9s 12ms/step - loss: 3.3585 - accu
racy: 0.2585 - val_loss: 3.6665 - val_accuracy: 0.2112
Epoch 17/20
782/782 [==============================] - 9s 12ms/step - loss: 3.2818 - accu
racy: 0.2722 - val_loss: 3.6654 - val_accuracy: 0.2122
Epoch 18/20
782/782 [==============================] - 9s 12ms/step - loss: 3.2148 - accu
racy: 0.2823 - val_loss: 3.6430 - val_accuracy: 0.2138
Epoch 19/20
782/782 [==============================] - 9s 12ms/step - loss: 3.1446 - accu
racy: 0.2940 - val_loss: 3.5986 - val_accuracy: 0.2234
Epoch 20/20
782/782 [==============================] - 9s 12ms/step - loss: 3.0798 - accu
racy: 0.3053 - val_loss: 3.6327 - val_accuracy: 0.2232
```

In [ ]:

```
# TODO: Train your model with 3 different numbers of epochs
batch_size = 32

# Setup your datasets
# TODO: Your Code Here
retrain = ds_retrain.batch(batch_size, num_parallel_calls=tf.data.AUTOTUNE)
reval = ds_reval.batch(batch_size, num_parallel_calls=tf.data.AUTOTUNE)

for epoch_size in [3, 10, 100]:
    # Run training
    # TODO: Your Code Here
    train_model.set_weights(weights=weights)
    train_model.fit(x=retrain, validation_data=reval, epochs=epoch_size, call

    # Save the cnn model
    train_model.save(os.path.join(log_dir, f'CNN_TinyImageNet_train_epoch{epo

    # TODO: Get the top-1 and top-5 of your newly trained model
    total = acc_top1 = acc_top5 = 0

    # TODO: Your Code Here
    for image in ds_val.batch(batch_size):
        prediction = train_model.predict(image['image'].numpy())

        top_1 = tf.math.top_k(prediction, k=1).indices
        top_5 = tf.math.top_k(prediction, k=5).indices

        for index, labels in enumerate(image['label'].numpy()):
            if labels in top_1[index]:
                acc_top1 += 1
            if labels in top_5[index]:
                acc_top5 += 1

        total += batch_size

    print(f"Top 1: {(acc_top1 / total)* 100:.2f}%, Top 5: {(acc_top5 / total)
```

```
Epoch 1/3
3125/3125 [==============================] - 17s 5ms/step - loss: 5.2374 - ac
curacy: 0.0088 - val_loss: 5.0587 - val_accuracy: 0.0197
Epoch 2/3
3125/3125 [==============================] - 16s 5ms/step - loss: 4.8771 - ac
curacy: 0.0401 - val_loss: 4.6827 - val_accuracy: 0.0629
Epoch 3/3
3125/3125 [==============================] - 16s 5ms/step - loss: 4.4578 - ac
curacy: 0.0912 - val_loss: 4.2967 - val_accuracy: 0.1106
Top 1: 8.11%, Top 5: 14.42%
Epoch 1/10
3125/3125 [==============================] - 17s 5ms/step - loss: 5.1722 - ac
curacy: 0.0146 - val_loss: 4.9507 - val_accuracy: 0.0316
Epoch 2/10
3125/3125 [==============================] - 16s 5ms/step - loss: 4.7465 - ac
curacy: 0.0555 - val_loss: 4.5753 - val_accuracy: 0.0793
Epoch 3/10
3125/3125 [==============================] - 16s 5ms/step - loss: 4.3093 - ac
curacy: 0.1093 - val_loss: 4.1985 - val_accuracy: 0.1167
Epoch 4/10
3125/3125 [==============================] - 17s 5ms/step - loss: 4.0029 - ac
```

```
curacy: 0.1512 - val_loss: 3.9361 - val_accuracy: 0.1601
Epoch 5/10
3125/3125 [==============================] - 17s 5ms/step - loss: 3.7801 - ac
curacy: 0.1838 - val_loss: 3.7881 - val_accuracy: 0.1826
Epoch 6/10
3125/3125 [==============================] - 16s 5ms/step - loss: 3.6053 - ac
curacy: 0.2126 - val_loss: 3.6934 - val_accuracy: 0.1981
Epoch 7/10
3125/3125 [==============================] - 17s 5ms/step - loss: 3.4610 - ac
curacy: 0.2356 - val_loss: 3.5876 - val_accuracy: 0.2141
Epoch 8/10
3125/3125 [==============================] - 16s 5ms/step - loss: 3.3265 - ac
curacy: 0.2578 - val_loss: 3.5800 - val_accuracy: 0.2192
Epoch 9/10
3125/3125 [==============================] - 16s 5ms/step - loss: 3.2068 - ac
curacy: 0.2798 - val_loss: 3.5652 - val_accuracy: 0.2233
Epoch 10/10
3125/3125 [==============================] - 16s 5ms/step - loss: 3.0987 - ac
curacy: 0.2981 - val_loss: 3.5036 - val_accuracy: 0.2315
Top 1: 11.56%, Top 5: 18.27%
Epoch 1/100
3125/3125 [==============================] - 17s 5ms/step - loss: 5.1788 - ac
curacy: 0.0130 - val_loss: 5.0259 - val_accuracy: 0.0238
Epoch 2/100
3125/3125 [==============================] - 16s 5ms/step - loss: 4.8665 - ac
curacy: 0.0434 - val_loss: 4.7043 - val_accuracy: 0.0642
Epoch 3/100
3125/3125 [==============================] - 17s 5ms/step - loss: 4.4663 - ac
curacy: 0.0912 - val_loss: 4.2927 - val_accuracy: 0.1127
Epoch 4/100
3125/3125 [==============================] - 16s 5ms/step - loss: 4.1179 - ac
curacy: 0.1381 - val_loss: 4.0086 - val_accuracy: 0.1508
Epoch 5/100
3125/3125 [==============================] - 16s 5ms/step - loss: 3.8488 - ac
curacy: 0.1769 - val_loss: 3.8156 - val_accuracy: 0.1808
Epoch 6/100
3125/3125 [==============================] - 17s 5ms/step - loss: 3.6498 - ac
curacy: 0.2077 - val_loss: 3.6929 - val_accuracy: 0.1982
Epoch 7/100
3125/3125 [==============================] - 17s 5ms/step - loss: 3.4875 - ac
curacy: 0.2334 - val_loss: 3.6326 - val_accuracy: 0.2098
Epoch 8/100
3125/3125 [==============================] - 17s 5ms/step - loss: 3.3483 - ac
curacy: 0.2579 - val_loss: 3.5373 - val_accuracy: 0.2267
Epoch 9/100
3125/3125 [==============================] - 17s 5ms/step - loss: 3.2238 - ac
curacy: 0.2782 - val_loss: 3.5222 - val_accuracy: 0.2310
Epoch 10/100
3125/3125 [==============================] - 16s 5ms/step - loss: 3.1063 - ac
curacy: 0.2990 - val_loss: 3.4915 - val_accuracy: 0.2378
Epoch 11/100
3125/3125 [==============================] - 17s 5ms/step - loss: 2.9929 - ac
curacy: 0.3198 - val_loss: 3.4371 - val_accuracy: 0.2469
Epoch 12/100
3125/3125 [==============================] - 16s 5ms/step - loss: 2.8854 - ac
curacy: 0.3391 - val_loss: 3.4653 - val_accuracy: 0.2505
Epoch 13/100
3125/3125 [==============================] - 17s 5ms/step - loss: 2.7797 - ac
curacy: 0.3568 - val_loss: 3.4933 - val_accuracy: 0.2475
Epoch 00013: early stopping
```

```
Top 1: 14.82%, Top 5: 21.20%
```

Above and Beyond

In [ ]:

```
# Benchmark our dataset to make sure loading our data isn't a bottleneck ...
# (This can be skipped since it can take a bit and is't all that important)

tfds.benchmark(ds_train.batch(32), batch_size=32, num_iter=2**20)
```

```
WARNING:absl:Number of iteration shorter than expected (3123 vs 1048576)
************ Summary ************

Examples/sec (First included) 140740.91 ex/sec (total: 100000 ex, 0.71 sec)
Examples/sec (First only) 2236.96 ex/sec (total: 32 ex, 0.01 sec)
Examples/sec (First excluded) 143586.72 ex/sec (total: 99968 ex, 0.70 sec)
```

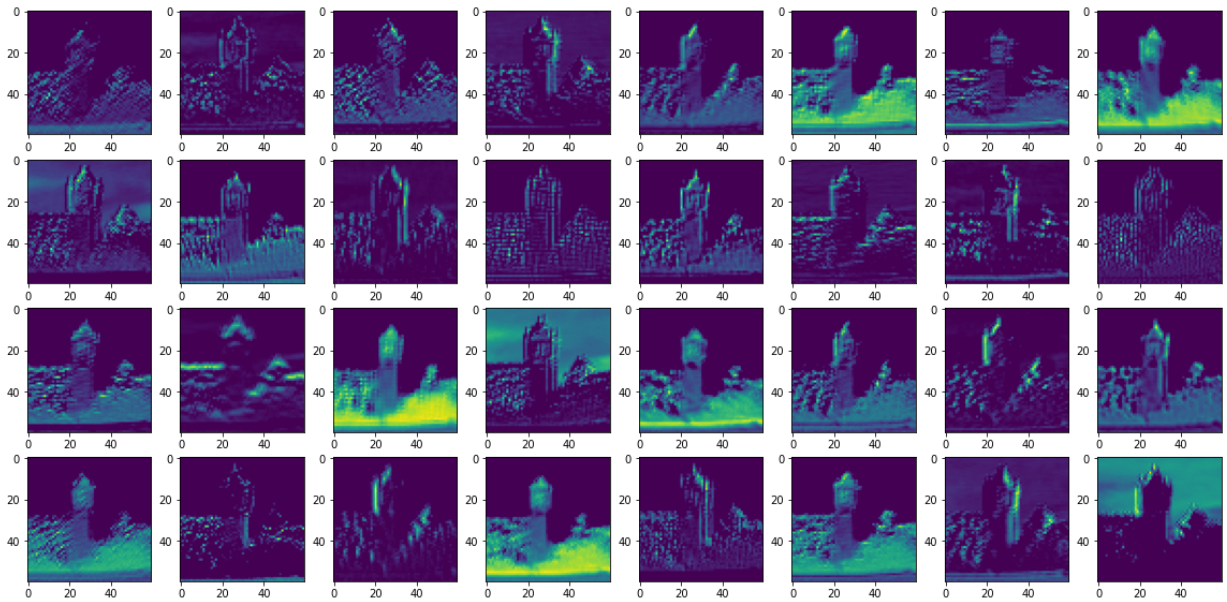
Out[ ]: **BenchmarkResult:**

	duration	num_examples	avg
first+lasts	0.710525	100000	140740.911071
first	0.014305	32	2236.964900
lasts	0.696220	99968	143586.723272

In [ ]:

```
# RUN INFERENCE ON A CUSTOM IMAGE AND DISPLAY FEATURE MAPS OF THE CAMPANILE!

# We can again view the layer outputs as well. Here we are pretending they ar
# TODO: Visualize the 2 convolutional layers outputs (intermediate feature ma


# TODO: Your Code Here

# Run inference to get feature maps on earlier layer
sub_model = Model(inputs=model.inputs, outputs=model.layers[0].output)

image = tf.keras.preprocessing.image.load_img('campanile.jpg', target_size=(6
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)
image = tf.keras.applications.vgg16.preprocess_input(image)

feature_maps = sub_model.predict(image)
idx = 1
fig = plt.figure(figsize=(20, 20))
for i in range (1, feature_maps.shape[3]+1):

    if idx == 65:
        break

    plt.subplot(8, 8, i)
    plt.imshow(feature_maps[0, :, :, i-1])
    idx += 1


plt.show()

print(f'Beginning Layer: {model.layers[0].name}') # Prove name of layer
print(f'Data Type: {feature_maps.dtype}') # Print dtype
print(f'Dimensions: {feature_maps.ndim} total {feature_maps.shape}') # Print
print(f'Required memory: {feature_maps.nbytes}B ({feature_maps.nbytes/1000}KB

# Run infrence to get feature maps on end conv layer

sub_model = Model(inputs=model.input, outputs=model.layers[7].output)

#print(f"Printing feature maps for layer {model.layers[7].name}...")

image = sample_imgs[0]['image']
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)
image = tf.keras.applications.vgg16.preprocess_input(image)

feature_maps = sub_model.predict(image)

idx = 1
fig = plt.figure(figsize=(20, 20))
for i in range (1, feature_maps.shape[3]+1):

    if idx == 65:
        break

    plt.subplot(8, 8, i)
    plt.imshow(feature_maps[0, :, :, i-1])
```

```
    idx += 1

plt.show()

print(f'End Layer: {model.layers[7].name}') # Prove name of layer
print(f'Data Type: {feature_maps.dtype}') # Print dtype
print(f'Dimensions: {feature_maps.ndim} total {feature_maps.shape}') # Print
print(f'Required memory: {feature_maps.nbytes}B ({feature_maps.nbytes/1000}KB
```



```
Beginning Layer: conv2d
Data Type: float32
Dimensions: 4 total (1, 60, 60, 32)
Required memory: 460800B (460.8KB)
```