CS5489 - Machine Learning

Lecture 10b - Deep Learning

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

- Going deeper
  - ReLU and Batchnorm
- Optimization methods
- **Deep architectures and Image classification**
- Transfer learning

Image Classification

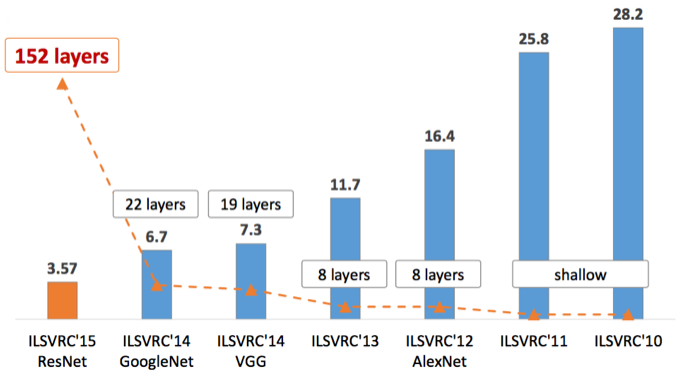
- **Goal:** given an image, predict the object class
  - typically only one main object is present
  - If enough classes are considered, then it's a generic high-level vision task

ImageNet

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - 1000 image classes
  - 1.2 million images
  - Human performance: ~5.1%

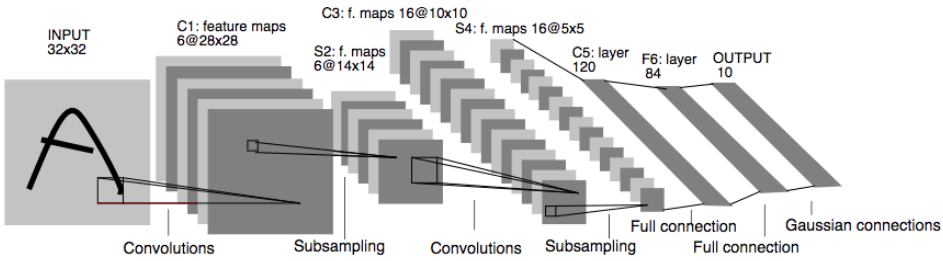
Performance of Deep Learning

- The introduction of ILSVRC coincided with the emergence of Deep Learning.
- Top-5 error rate decreased as deeper NN were developed
  - Not just deeper, but also the architecture design was smarter



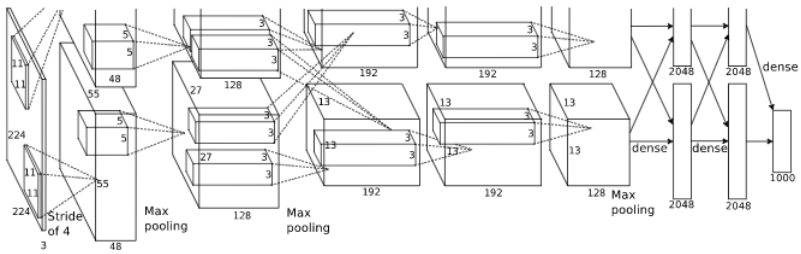
LeNet-5 (1998)

- The standard CNN architecture
  - 7 layers
  - convolutions & pooling, final fully-connected layer
- Designed for hand-written digit recongition (MNIST)



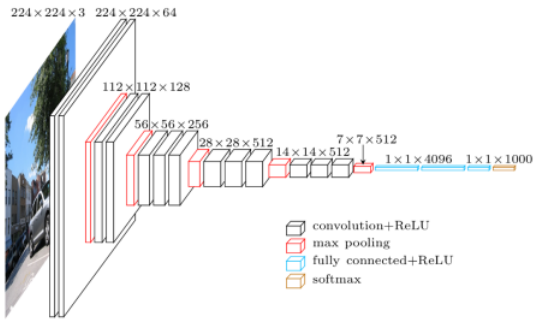
AlexNet

- Similar architecture to LeNet, but deeper (14 layers)
  - 11x11, 5x5, 3x3 convolutions
- Tricks used: DropOut, ReLU, max pooling, data augmentation, SGD momentum
- One of the first networks trained on GPU
  - (it is split in two pipelines because it was trained on 2 GPUs simultaneously)



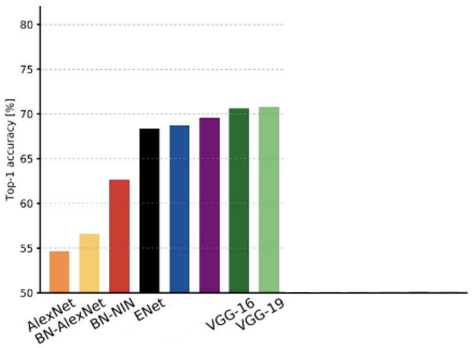
VGGnet

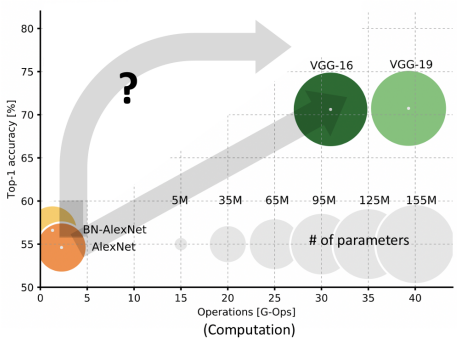
- Same style as LeNet and AlexNet
- Design choices:
  - only use 3x3 convolution filters (less parameters)
  - stack several convolution layers one after another, followed by pooling
    - equivalent to using a larger filter, but with less parameters.
  - number of feature channels doubles after each stage.
    - more higher-level features



Progress so far...

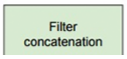
- Deeper...more parameters...better accuracy...
  - *performance is saturating*





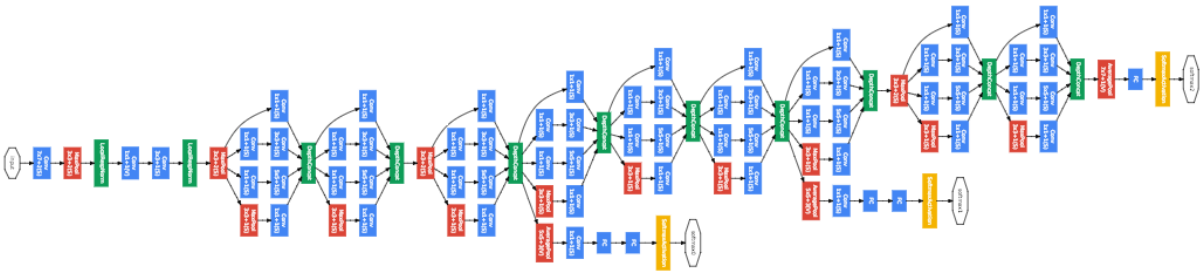
InceptionNet

- "Network-in-Network" architecture
  - Core building blocks (modules), consisting of several layers, are combined repeatedly.
- Inception Module
  - several convolution filters in parallel
    - extract features at different scales (1x1, 3x3, 5x5)
    - pool features (3x3 max)
  - features are concatenated and passed to next block.



InceptionNet (V1)

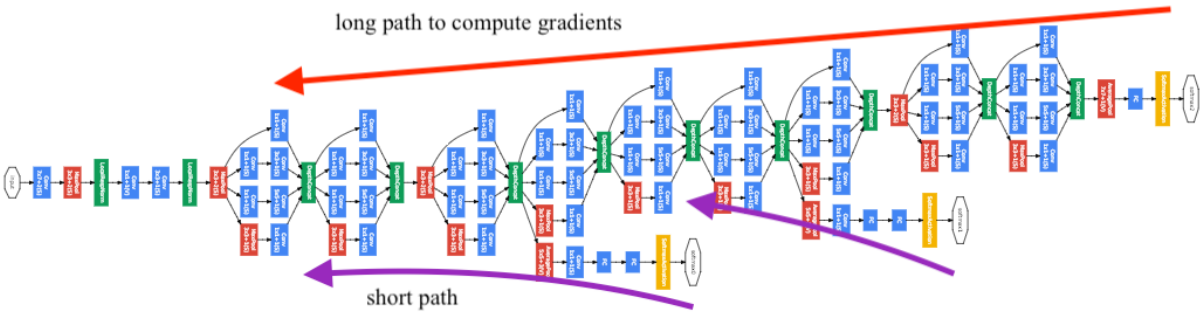
- 9 inception modules, 22 layers
  - 50 convolution blocks
- Auxiliary classification tasks
  - using features in the middle of the network to perform classification (yellow boxes)



Auxiliary tasks

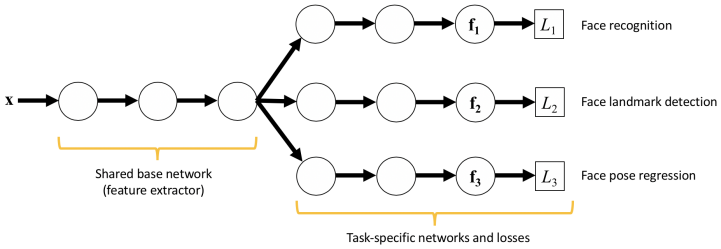
- **Goal:** strengthen the supervisory signal to the early layers
- perform the same classification task with branches off the main network.
- new loss is sum of individual classification losses:
  - $L = \lambda_0 L_0 + \lambda_1 L_1 + \lambda_2 L_2$
  - $\frac{dL}{d\mathbf{w}} = \lambda_0 \frac{dL_0}{d\mathbf{w}} + \lambda_1 \frac{dL_1}{d\mathbf{w}} + \lambda_2 \frac{dL_2}{d\mathbf{w}}$

- auxiliary tasks shorten the path to compute the gradients of early layers.
  - more direct supervision to the earlier layers.
  - encourages earlier layers to be more discriminative.

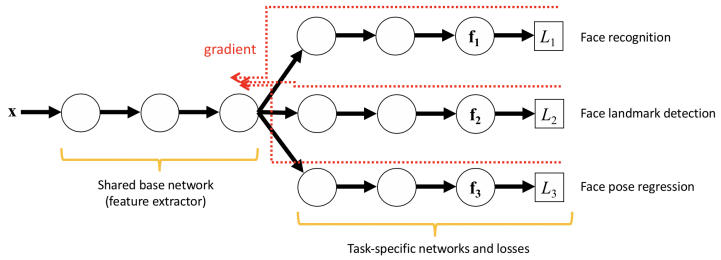


Multi-task learning

- several related tasks are learned together using a shared base network.



- multiple supervisory signals make the common network find good features for all tasks.
  - gradients are accumulated in the common base network.



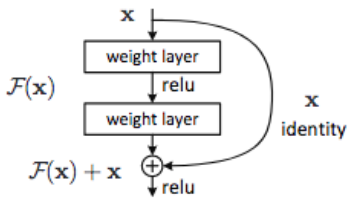
- if the tasks are related, the features for one task could be useful for the other tasks.
  - e.g., facial landmark locations could help face recognition.

Going deeper

- **Idea:** when stacking more layers, we should not have degraded performance, since we could make the new layer as an "identity" layer.
  - if $\mathbf{A} = \mathbf{I}$, then $f_i(\mathbf{x}) = \mathbf{A}^T\mathbf{x} = \mathbf{x}$
  - i.e., this layer doesn't change the hidden values from the previous layer.
- **Problem:** if we use L2 regularization on $\mathbf{A}$, then we encourage $\mathbf{A} \to 0$.
  - so we cannot get this behavior...

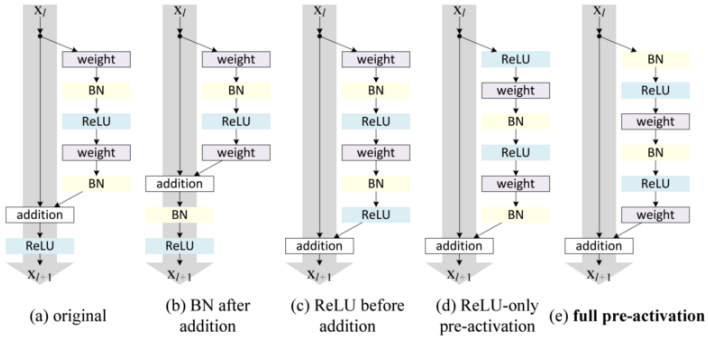
Residual Learning

- **Solution**: learn a residual function so that if $\mathbf{A} = 0$, then $f(\mathbf{x}) = \mathbf{x}$
  - residual function $\mathcal{F}(\mathbf{x}) = \mathbf{A}^T\mathbf{x}$
  - output: $f(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$
    - learn the $\mathbf{A}$ in the residual.
    - when $\mathbf{A} = 0$, then $f(\mathbf{x}) = \mathbf{x}$.
  - the "identity path" is sometimes called a "shortcut" connection.



Residual Blocks

- Different versions of the blocks.
  - differences in where to add the identity
  - (e) is preferred.

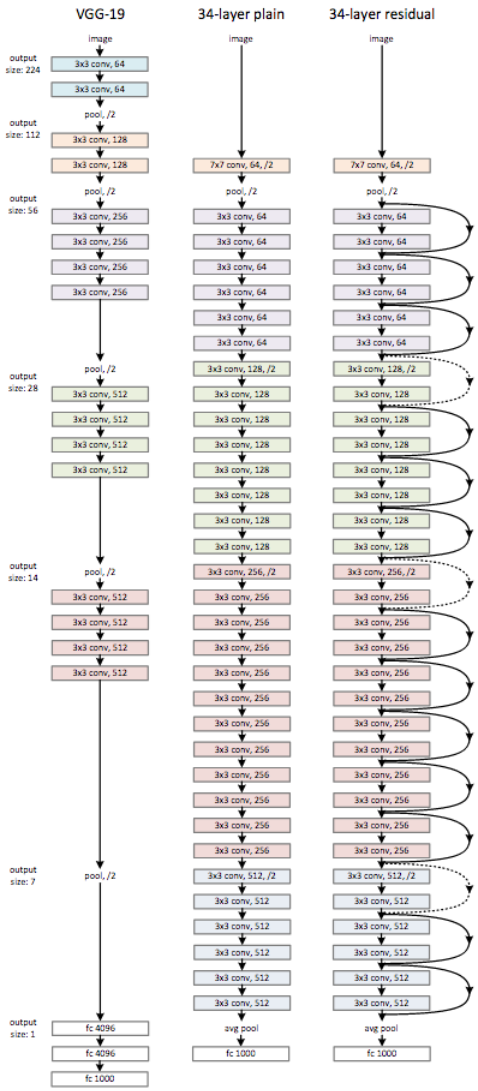


Stacking Residual Blocks

- The network is learning a function (image to class)
  - build the function by appending residual blocks
  - each block learns a residual, which is added to the previous block
    - keep all the previous information and make small changes with the residual

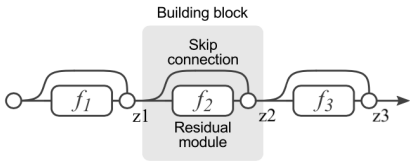
Residual Network (ResNet)

- 34 layers, 50 layers, 100 layers, 1000 layers
  - 3x3 filters
  - residual connection every two layers.



ResNet as an Ensemble

- Consider a 3-block ResNet



- Look at the equations:

$$\begin{aligned}\mathbf{z}_1 &= \mathbf{x} + f_1(\mathbf{x}),\\ \mathbf{z}_2 &= \mathbf{z}_1 + f_2(\mathbf{z}_1),\\ \mathbf{z}_3 &= \mathbf{z}_2 + f_3(\mathbf{z}_2)\end{aligned}$$

- Substitute for $\mathbf{z}_2$...

$$\mathbf{z}_3 = \underbrace{\mathbf{z}_1 + f_2(\mathbf{z}_1)}_{} + f_3(\underbrace{\mathbf{z}_1 + f_2(\mathbf{z}_1)}_{})$$

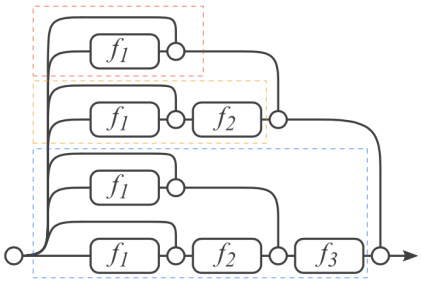
- Substitute for $\mathbf{z}_1$...

$$\mathbf{z}_3 = \underbrace{\mathbf{x} + f_1(\mathbf{x})}_{} + f_2(\underbrace{\mathbf{x} + f_1(\mathbf{x})}_{}) + f_3(\underbrace{\mathbf{x} + f_1(\mathbf{x})}_{} + f_2(\underbrace{\mathbf{x} + f_1(\mathbf{x})}_{}))$$

- the output is sum of different combinations of residual blocks

$$\mathbf{z}_3 = \underbrace{\mathbf{x} + f_1(\mathbf{x})}_{} + \underbrace{f_2(\underbrace{\mathbf{x} + f_1(\mathbf{x})}_{})}_{} + \underbrace{f_3(\underbrace{\mathbf{x} + f_1(\mathbf{x})}_{} + \underbrace{f_2(\underbrace{\mathbf{x} + f_1(\mathbf{x})}_{})}_{})}_{}$$

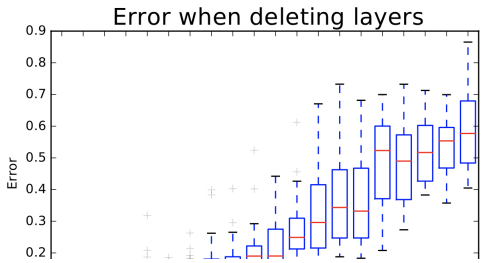
  - equivalent to an *ensemble*!



- *How many paths are there?*
  - can select block f_i or not, thus 2^N paths.
    - where N is the number of residual blocks.
  - 2^N models in the ensemble

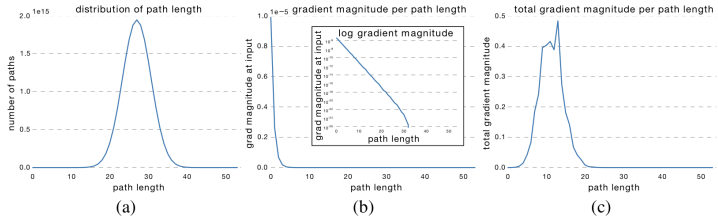
Robustness

- Because it is an ensemble, ResNet is robust to removal of a few blocks (sub-models).
  - error is still small when a few blocks are removed.



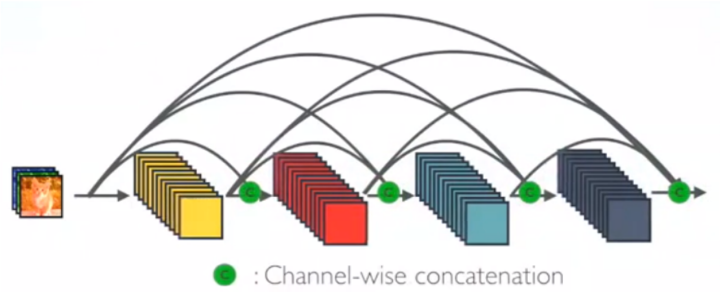
Paths and Gradients

- (a) the paths through the ResNet has different lengths
  - number of paths with length L = "N choose L", N = number of blocks.
- (b) shorter paths have more effective gradients because they have direct supervision from the loss.
  - reduces the vanishing gradient problem.
- (c) although there are many paths, the short paths contain most of the gradient signal.
  - effectively, the network is shallow during training.



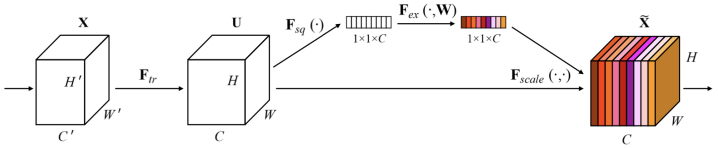
DenseNet

- Each layer gets inputs from all previous layers
  - Since features maps are passed forward, fewer output channels are required in each layer.
  - last layer uses all previous feature maps (of various semantic levels).



Attention Mechanisms

- ignore unimportant features (channels or spatial regions) in the feature map by using a (soft) mask.
- i.e., attend to only important features (or spatial regions)
- "Squeeze-Excitation Block"
  - "squeeze" feature map into 1x1 channel vector
  - "excite" features by scaling up important ones, and scaling down less-important ones.



SEnet

- the SE block can be incorporated into standard architectures.

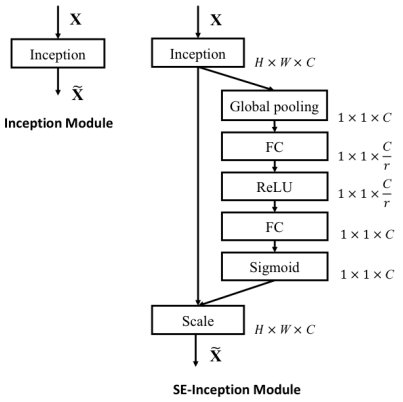


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

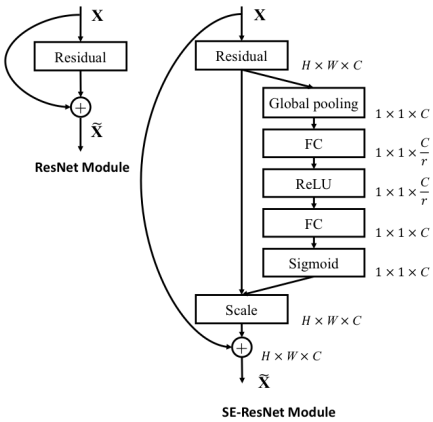
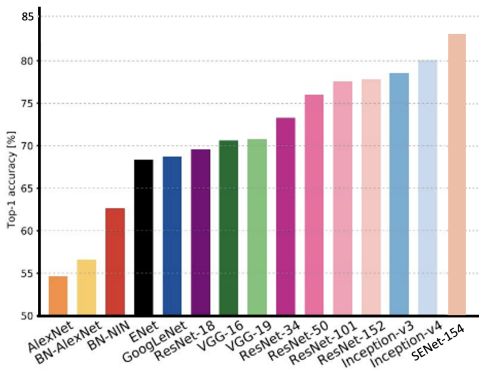
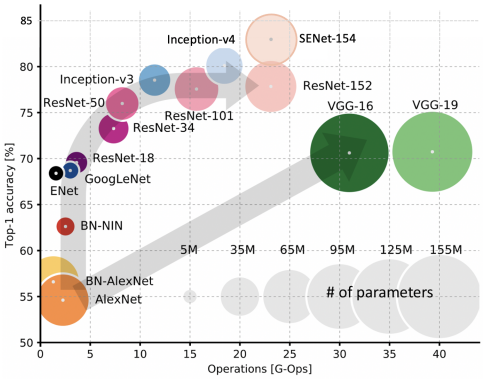


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

Error vs. Computation vs. Number of Parameters

- Just adding more layers is limited.
  - Too many parameters
- More improvement by changing the architecture design.
  - Smarter use of the parameters
  - **Architecture Design is important!**





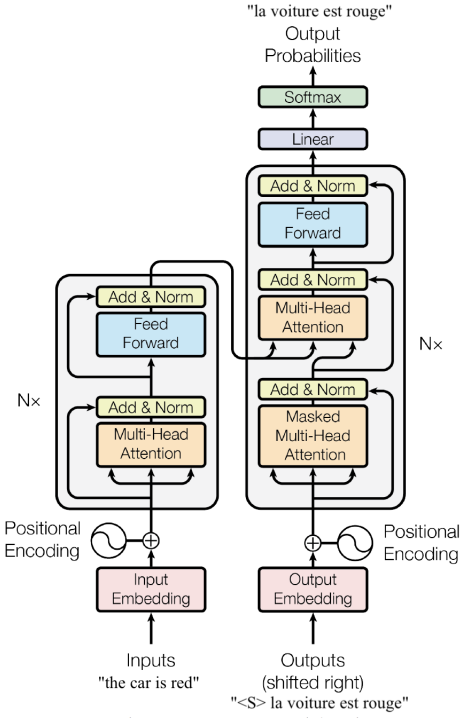
Moving forwards

- Performance with CNN-style architectures seems to be saturated.
- **Advantage:**
  - inductive bias (assumptions) of CNNs: translation equivariance, locality
- **Disadvantages:**
  - cannot well model long-range dependencies (global context)
    - limited receptive field means 2 far-apart pixels are connected via a long path.
  - low speed - adding more layers cannot be parallelized.
- **Solution:** need a new type of architecture.

Transformers

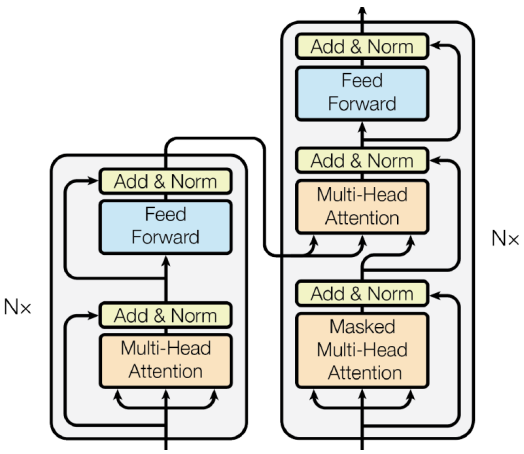
- NLP sequence-to-sequence model (e.g. machine translation)
  - remove previous inductive biases: no recurrences, no convolutions

- processes all words (inputs) in parallel
- selectively attend to different locations with a *self-attention* mechanism

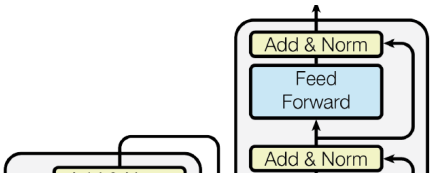


Encoder/Decoder

- Stack of 6 blocks with identical architecture
  - multi-head self-attention
  - residual connection
  - layer normalization: normalize all features within layer to N(0,1)
  - position-wise feed-forward NN (MLP): identical fo reach position.

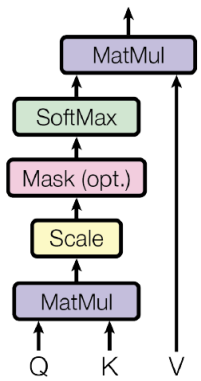


- Decoder:
  - adds mask to MHA to predict the current word only using previous words.
  - apply MHA on output of the encoder.



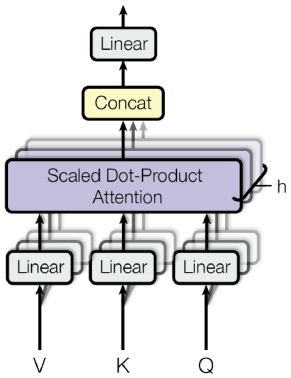
Dot-Product Attention

- Attention mechanism has 3 inputs:
  - query $\mathbf{Q} \in \mathbb{R}^{d_k \times N}$: what we want to match.
  - key $\mathbf{K} \in \mathbb{R}^{d_k \times N}$: what we match against.
  - value $\mathbf{V} \in \mathbb{R}^{d_v \times N}$: what to output when there is a match.
  - d_k, d_v are the dimensions of the key-query space and value space.
  - N is the number of inputs (positions).
- Attention matrix:
  - $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_N] = \mathrm{softmax}(\frac{1}{\sqrt{d_k}}\mathbf{K}^T\mathbf{Q}) \in \mathbb{R}^{N \times N}$
    - $\mathbf{a}_j = \mathrm{softmax}(\frac{1}{\sqrt{d_k}}\mathbf{K}^T\mathbf{q}_j)$ is the vector of matches for the j-th query (to all keys).
    - A_{ij} is the match between the j-th query and the i-th key.
- Output: $\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{V}\mathbf{A} = [\mathbf{V}\mathbf{a}_1 \cdots \mathbf{V}\mathbf{a}_D]$
  - for the j-th query, linear combination of values for matched keys.

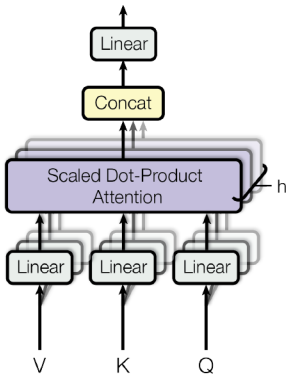


Multi-Head Attention

- Input: $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\} \in \mathbb{R}^{D \times N}$
- Perform multiple dot-product attentions in parallel
- Each *head* projects the input $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$ to its own lower-dim key/query/value space that is learned
  - $\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d_k \times D}, \mathbf{W}_i^V \in \mathbb{R}^{d_v \times D}$
  - $\mathbf{Q}' = \mathbf{W}_i^Q\mathbf{Q}, \mathbf{K}' = \mathbf{W}_i^K\mathbf{K}, \mathbf{V}' = \mathbf{W}_i^V\mathbf{V}$
  - $\mathbf{H}_i = \mathrm{Attn}(\mathbf{Q}', \mathbf{K}', \mathbf{V}')$

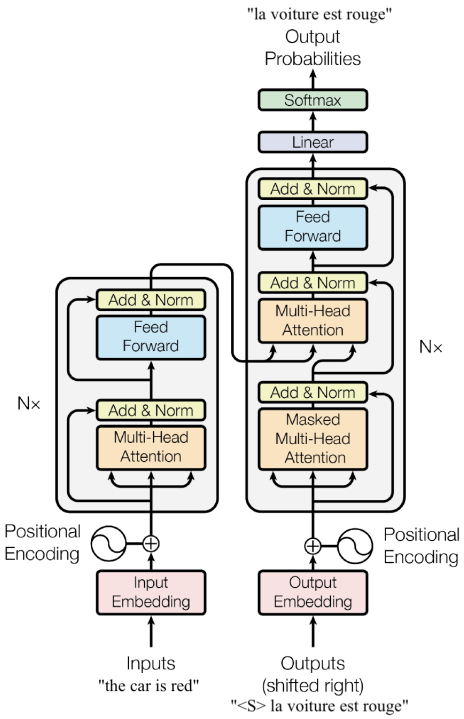


- Output: concatenate head outputs and project to original space
  - $\mathrm{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{W}^O \begin{bmatrix} \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_h \end{bmatrix}$
  - $\mathbf{W}^O \in \mathbb{R}^{D \times h d_v}$
- Example dimensions:
  - number of heads $h = 8$, embedding dimension $D = 512$
  - $d_k = d_v = D/h = 64$

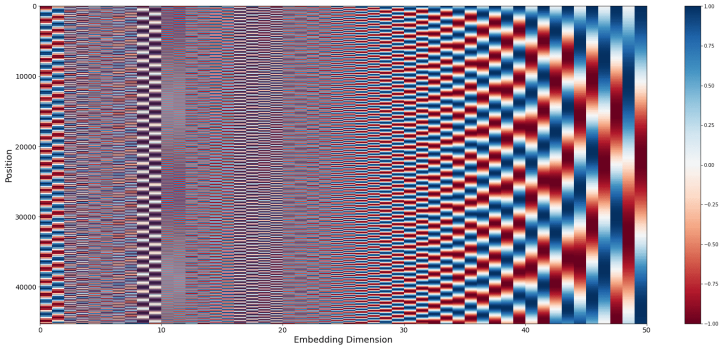


Other things

- position-wise NN (2-layer MLP) applies to each position at end of each block.
- learnable input embeddings: turn each word (token) into a vector.
  - $\mathbf{x}' = \mathbf{A}\mathbf{x}$
    - $\mathbf{x}$ is a one-hot encoding of the word.
    - $\mathbf{A}$ is a matrix of embedding vectors.

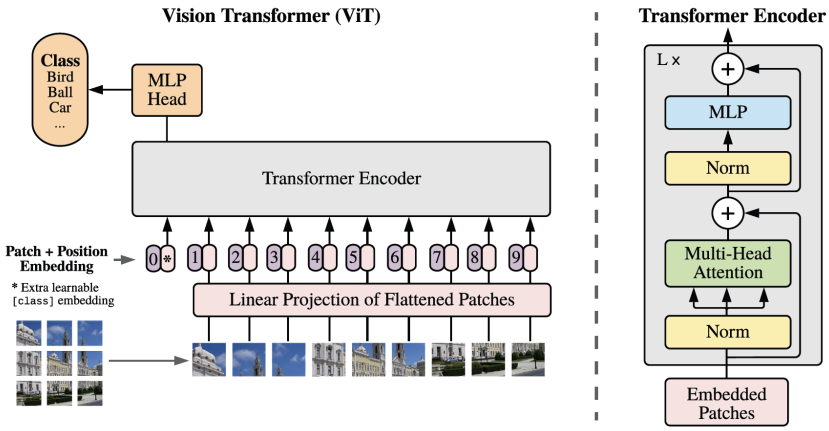


- position-encoding
  - the relative positions between inputs is not built into the self-attention architecture (c.f., convolution layer)
  - add a position-embedding to each word vector at the input.
  - learnable or fixed (sine and cosine waves of different frequencies)

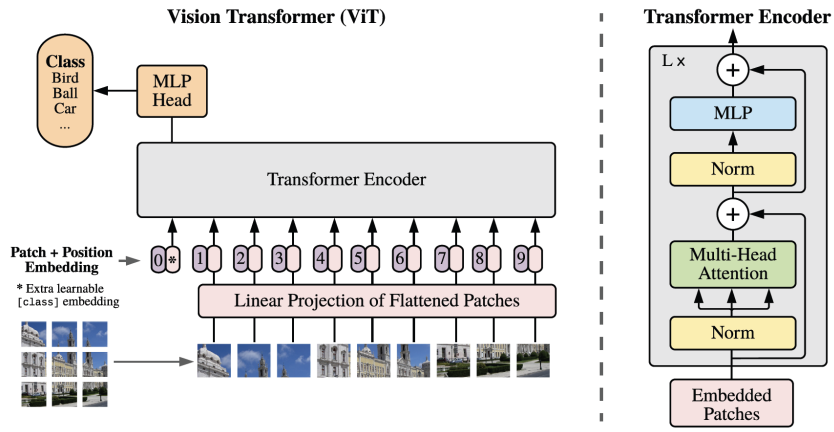


Vision Transformers (ViT)

- Apply transformer architecture to image classification
  1. Split image into patches (16x16 pixels).
  2. Flatten and then embed with linear projection.
  3. Prepend a learnable "[class]" token.



- Apply transformer architecture to image classification 3. Add position embeddings 4. Apply transformer encoder 5. Classification head: MLP on the first output of the encoder.



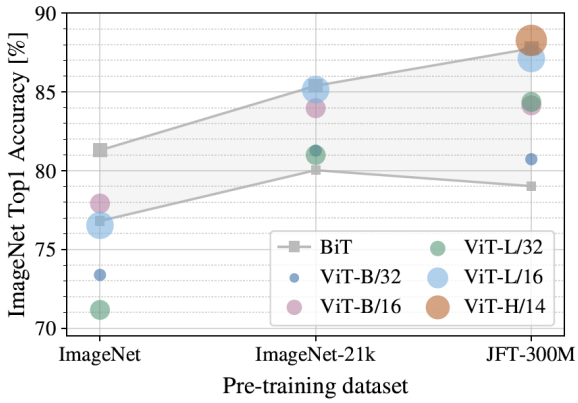
Model variations

- Different variants with different depth.
- Can also apply ViT to the feature map from a CNN (Hybrid model)

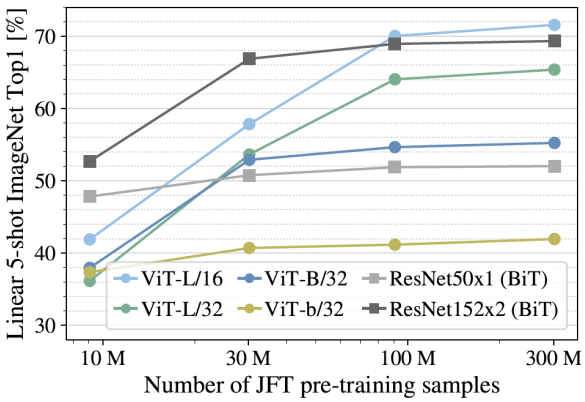
Model	Layers	Hidden size D	MLP size	Heads	Params

Model Pre-training & Results

- Pre-training on different datasets, test on ImageNet1k
  - worse than ResNets when trained on IN1k (1k classes, 1.3M images)
  - similar when trained on IN21k (21k classes, 14M images)
  - better when trained on JFT (18k classes, 303M images)
  - **Why?**

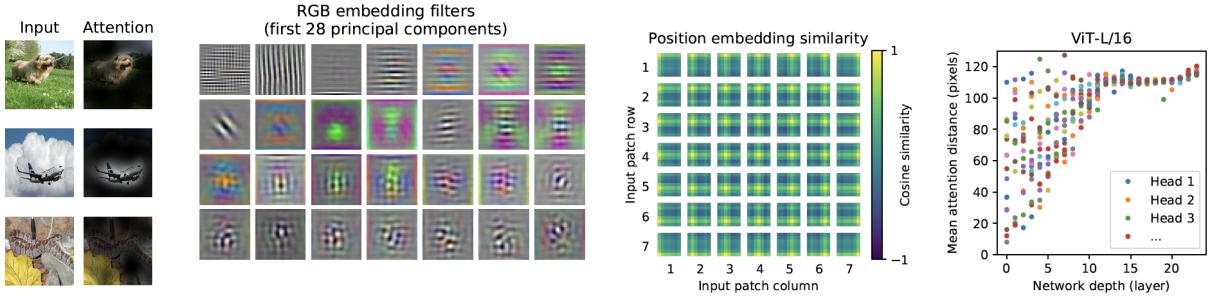


- **Reason:**
  - The inductive biases in ResNet helps on smaller datasets (IN1k), but ViT does not have enough data to learn from.
  - Large scale training on JFT, then ViT can outperform the inductive bias.



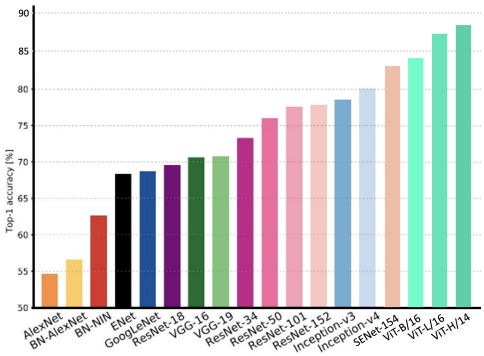
Visualization & Intepretation

- Visualization of the models attention, linear embeddings, position encoding, and attention head



Error vs. Computation vs. Number of Parameters

- Transformer and self-attention mechanism
  - improves performance
  - lots of parameters
  - lots of computation (quadratic in number of patches).



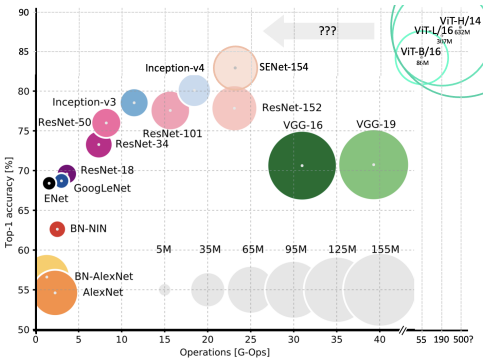
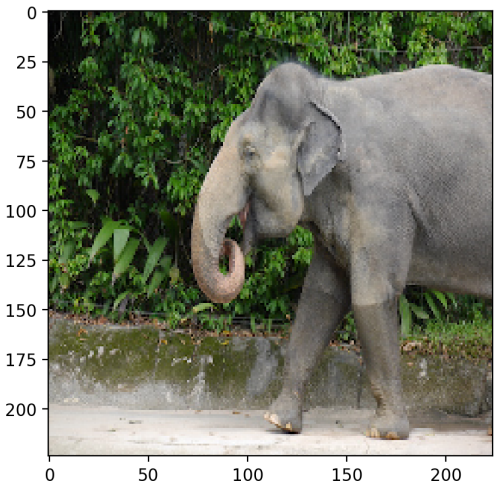


Image Classifiers in Keras

- Keras includes several pre-trained image classifier networks
  - VGG16, VGG19, ResNet50, DenseNet, InceptionV3, ...
  - already trained on ImageNet
- Can use these networks to:
  - perform image classification (for 1000 classes only)
  - extract image features for our own task
  - transfer learning - modify/adapt a pre-trained network for our own task

- Import the ResNet model class and support functions
- Load and pre-process the image for the network

In [6]:
```
# contains the model and support functions for ResNet50
import tensorflow.keras.applications.resnet50 as resnet
from tensorflow.keras.preprocessing import image

# load an image
img_path = 'imgs/elephant1.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # ResNet expects this size
plt.imshow(img)

# process the image for resnet
xi = image.img_to_array(img)
xi = expand_dims(xi, axis=0)
xi = resnet.preprocess_input(xi)
```



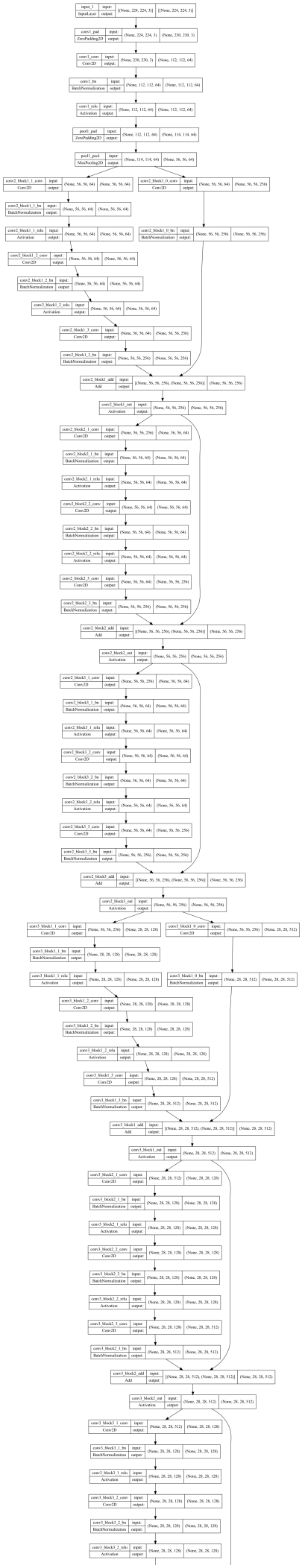
- Create an instance of the ResNet50 model
  - trained on ImageNet
  - will download the model if loading for the first time.

In [7]:
```
# create an instance of the model
model = resnet.ResNet50(weights='imagenet')
```

```
2023-01-23 16:00:33.722315: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-23 16:00:33.722335: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
```

In [8]:
```
# visualize the network
tmp = tf.keras.utils.plot_model(model, to_file='tmp_model.png', show_shapes=True)
tmp.width=400
tmp
```

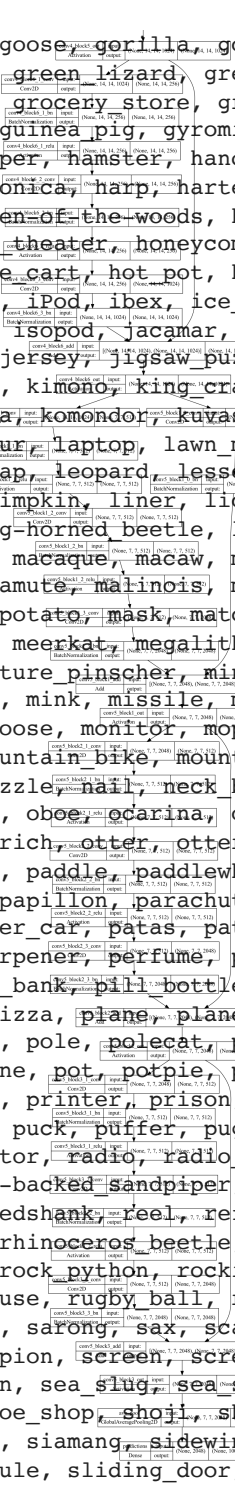
Out[8]:


input_1 InputLayer
conv1_pad ZeroPadding2D
conv1_conv Conv2D
conv1_bn BatchNormalization
conv1_relu Activation
pool1_pad ZeroPadding2D
pool1_pool MaxPooling2D
conv2_block1_1_conv Conv2D
conv2_block1_0_conv Conv2D
conv2_block1_1_bn BatchNormalization
conv2_block1_1_relu Activation
conv2_block1_0_bn BatchNormalization
conv2_block1_2_conv Conv2D
conv2_block1_2_bn BatchNormalization
conv2_block1_2_relu Activation
conv2_block1_3_conv Conv2D
conv2_block1_3_bn BatchNormalization
conv2_block1_add Add
conv2_block1_out Activation
conv2_block2_1_conv Conv2D
conv2_block2_1_bn BatchNormalization
conv2_block2_1_relu Activation
conv2_block2_2_conv Conv2D
conv2_block2_2_bn BatchNormalization
conv2_block2_2_relu Activation
conv2_block2_3_conv Conv2D
conv2_block2_3_bn BatchNormalization
conv2_block2_add Add
conv2_block2_out Activation
conv2_block3_1_conv Conv2D
conv2_block3_1_bn BatchNormalization
conv2_block3_1_relu Activation
conv2_block3_2_conv Conv2D
conv2_block3_2_bn BatchNormalization
conv2_block3_2_relu Activation
conv2_block3_3_conv Conv2D
conv2_block3_3_bn BatchNormalization
conv2_block3_add Add
conv2_block3_out Activation
conv3_block1_1_conv Conv2D
conv3_block1_0_conv Conv2D
conv3_block1_1_bn BatchNormalization
conv3_block1_1_relu Activation
conv3_block1_0_bn BatchNormalization
conv3_block1_2_conv Conv2D
conv3_block1_2_bn BatchNormalization
conv3_block1_2_relu Activation
conv3_block1_3_conv Conv2D
conv3_block1_3_bn BatchNormalization
conv3_block1_add Add
conv3_block1_out Activation
conv3_block2_1_conv Conv2D
conv3_block2_1_bn BatchNormalization
conv3_block2_1_relu Activation
conv3_block2_2_conv Conv2D
conv3_block2_2_bn BatchNormalization
conv3_block2_2_relu Activation
conv3_block2_3_conv Conv2D
conv3_block2_3_bn BatchNormalization
conv3_block2_add Add
conv3_block2_out Activation
conv3_block3_1_conv Conv2D
conv3_block3_1_bn BatchNormalization
conv3_block3_1_relu Activation
conv3_block3_2_conv Conv2D
conv3_block3_2_bn BatchNormalization
conv3_block3_2_relu Activation

- Look at the 1000 classes

In [9]:
```
tmp = resnet.decode_predictions(eye(1000), top=1)  # decode the 1000-dim class vector
tmp2 = [i[0][1] for i in tmp]
tmp2.sort()
print(", ".join(tmp2))
```

```
Afghan_hound, African_chameleon, African_crocodile, African_elephant, African_gre
y, African_hunting_dog, Airedale, American_Staffordshire_terrier, American_alligat
or, American_black_bear, American_chameleon, American_coot, American_egret, Americ
an_lobster, Angora, Appenzeller, Arabian_camel, Arctic_fox, Australian_terrier, Ba
nd_Aid, Bedlington_terrier, Bernese_mountain_dog, Blenheim_spaniel, Border_collie,
Border_terrier, Boston_bull, Bouvier_des_Flandres, Brabancon_griffon, Brittany_spa
niel, CD_player, Cardigan, Chesapeake_Bay_retriever, Chihuahua, Christmas_stockin
g, Crock_Pot, Dandie_Dinmont, Doberman, Dungeness_crab, Dutch_oven, Egyptian_cat,
English_foxhound, English_setter, English_springer, EntleBucher, Eskimo_dog, Europ
ean_fire_salamander, European_gallinule, French_bulldog, French_horn, French_loaf,
German_shepherd, German_short-haired_pointer, Gila_monster, Gordon_setter, Granny_
Smith, Great_Dane, Great_Pyrenees, Greater_Swiss_Mountain_dog, Ibizan_hound, India
n_cobra, Indian_elephant, Irish_setter, Irish_terrier, Irish_water_spaniel, Irish_
wolfhound, Italian_greyhound, Japanese_spaniel, Kerry_blue_terrier, Komodo_dragon,
Labrador_retriever, Lakeland_terrier, Leonberg, Lhasa, Loafer, Madagascar_cat, Mal
tese_dog, Mexican_hairless, Model_T, Newfoundland, Norfolk_terrier, Norwegian_elkh
ound, Norwich_terrier, Old_English_sheepdog, Pekinese, Pembroke, Persian_cat, Petr
i_dish, Polaroid_camera, Pomeranian, Rhodesian_ridgeback, Rottweiler, Saint_Bernar
d, Saluki, Samoyed, Scotch_terrier, Scottish_deerhound, Sealyham_terrier, Shetland
_sheepdog, Shih-Tzu, Siamese_cat, Siberian_husky, Staffordshire_bullterrier, Susse
x_spaniel, Tibetan_mastiff, Tibetan_terrier, Walker_hound, Weimaraner, Welsh_sprin
ger_spaniel, West_Highland_white_terrier, Windsor_tie, Yorkshire_terrier, abacus,
abaya, academic_gown, accordion, acorn, acorn_squash, acoustic_guitar, admiral, af
fenpinscher, agama, agaric, aircraft_carrier, airliner, airship, albatross, alliga
tor_lizard, alp, altar, ambulance, amphibian, analog_clock, anemone_fish, ant, api
ary, apron, armadillo, artichoke, ashcan, assault_rifle, axolotl, baboon, backpac
k, badger, bagel, bakery, balance_beam, bald_eagle, balloon, ballplayer, ballpoin
t, banana, banded_gecko, banjo, bannister, barbell, barber_chair, barbershop, bar
n, barn_spider, barometer, barracouta, barrel, barrow, baseball, basenji, basketba
ll, basset, bassinet, bassoon, bath_towel, bathing_cap, bathtub, beach_wagon, beac
on, beagle, beaker, bearskin, beaver, bee, bee_eater, beer_bottle, beer_glass, bel
l_cote, bell_pepper, bib, bicycle-built-for-two, bighorn, bikini, binder, binocula
rs, birdhouse, bison, bittern, black-and-tan_coonhound, black-footed_ferret, black
_and_gold_garden_spider, black_grouse, black_stork, black_swan, black_widow, blood
hound, bluetick, boa_constrictor, boathouse, bobsled, bolete, bolo_tie, bonnet, bo
ok_jacket, bookcase, bookshop, borzoi, bottlecap, bow, bow_tie, box_turtle, boxer,
brain_coral, brambling, brass, brassiere, breakwater, breastplate, briard, broccol
i, broom, brown_bear, bubble, bucket, buckeye, buckle, bulbul, bull_mastiff, bulle
t_train, bulletproof_vest, bullfrog, burrito, bustard, butcher_shop, butternut_squ
ash, cab, cabbage_butterfly, cairn, caldron, can_opener, candle, cannon, canoe, ca
puchin, car_mirror, car_wheel, carbonara, cardigan, cardoon, carousel, carpenter's
_kit, carton, cash_machine, cassette, cassette_player, castle, catamaran, cauliflo
wer, cello, cellular_telephone, centipede, chain, chain_mail, chain_saw, chainlink
_fence, chambered_nautilus, cheeseburger, cheetah, chest, chickadee, chiffonier, c
hime, chimpanzee, china_cabinet, chiton, chocolate_sauce, chow, church, cicada, ci
nema, cleaver, cliff, cliff_dwelling, cloak, clog, clumber, cock, cocker_spaniel,
cockroach, cocktail_shaker, coffee_mug, coffeepot, coho, coil, collie, colobus, co
mbination_lock, comic_book, common_iguana, common_newt, computer_keyboard, conch,
confectionery, consomme, container_ship, convertible, coral_fungus, coral_reef, co
rkscrew, corn, cornet, coucal, cougar, cowboy_boot, cowboy_hat, coyote, cradle, cr
ane, crane, crash_helmet, crate, crayfish, crib, cricket, croquet_ball, crossword_
puzzle, crutch, cucumber, cuirass, cup, curly-coated_retriever, custard_apple, dai
sy, dalmatian, dam, damselfly, desk, desktop_computer, dhole, dial_telephone, diam
ondback, diaper, digital_clock, digital_watch, dingo, dining_table, dishrag, dishw
asher, disk_brake, dock, dogsled, dome, doormat, dough, dowitcher, dragonfly, drak
e, drilling_platform, drum, drumstick, dugong, dumbbell, dung_beetle, ear, earthst
ar, echidna, eel, eft, eggnog, electric_fan, electric_guitar, electric_locomotive,
electric_ray, entertainment_center, envelope, espresso, espresso_maker, face_powde
r, feather_boa, fiddler_crab, fig, file, fire_engine, fire_screen, fireboat, flagp
ole, flamingo, flat-coated_retriever, flatworm, flute, fly, folding_chair, footbal
l_helmet, forklift, fountain, fountain_pen, four-poster, fox_squirrel, freight_ca
r, frilled_lizard, frying_pan, fur_coat, gar, garbage_truck, garden_spider, garter
_snake, gas_pump, gasmask, gazelle, geyser, giant_panda, giant_schnauzer, gibbon,
go-kart, goblet, golden_retriever, goldfinch, goldfish, golf_ball, golfcart, gondo
```



la, gong, goose, gorilla, gown, grand_piano, grasshopper, great_grey_owl, great_wh
ite_shark, green_lizard, green_mamba, green_snake, greenhouse, grey_fox, grey_whal
e, grille, grocery_store, groenendael, groom, ground_beetle, guacamole, guenon, gu
illotine, guinea_pig, gyromitra, hair_slide, hair_spray, half_track, hammer, hamme
rhead, hamper, hamster, hand-held_computer, hand_blower, handkerchief, hard_disc,
hare, harmonica, harp, hartebeest, harvester, harvestman, hatchet, hay, head_cabba
ge, hen, hen-of-the-woods, hermit_crab, hip, hippopotamus, hog, hognose_snake, hol
ster, home_theater, honeycomb, hook, hoopskirt, horizontal_bar, hornbill, horned_v
iper, horse_cart, hot_pot, hotdog, hourglass, house_finch, howler_monkey, hummingb
ird, hyena, iPod, ibex, ice_bear, ice_cream, ice_lolly, impala, indigo_bunting, in
dri, iron, isopod, jacamar, jack-o'-lantern, jackfruit, jaguar, jay, jean, jeep, j
ellyfish, jersey, jigsaw_puzzle, jinrikisha, joystick, junco, keeshond, kelpie, ki
ller_whale, kimono, king_crab, king_penguin, king_snake, kit_fox, kite, knee_pad,
knot, koala, komondor, kuvasz, lab_coat, lacewing, ladle, ladybug, lakeside, lamps
hade, langur, laptop, lawn_mower, leaf_beetle, leafhopper, leatherback_turtle, lem
on, lens_cap, leopard, lesser_panda, letter_opener, library, lifeboat, lighter, li
mousine, limpkin, liner, lion, lionfish, lipstick, little_blue_heron, llama, logge
rhead, long-horned_beetle, lorikeet, lotion, loudspeaker, loupe, lumbermill, lycae
nid, lynx, macaque, macaw, magnetic_compass, magpie, mailbag, mailbox, maillot, ma
illot, malamute, malinois, manhole_cover, mantis, maraca, marimba, marmoset, marmo
t, mashed_potato, mask, matchstick, maypole, maze, measuring_cup, meat_loaf, medic
ine_chest, meerkat, megalith, menu, microphone, microwave, military_uniform, milk_
can, miniature_pinscher, miniature_poodle, miniature_schnauzer, minibus, miniskir
t, minivan, mink, missile, mitten, mixing_bowl, mobile_home, modem, monarch, monas
tery, mongoose, monitor, moped, mortar, mortarboard, mosque, mosquito_net, motor_s
cooter, mountain_bike, mountain_tent, mouse, mousetrap, moving_van, mud_turtle, mu
shroom, muzzle, nail, neck_brace, necklace, nematode, night_snake, nipple, noteboo
k, obelisk, oboe, ocarina, odometer, oil_filter, orange, orangutan, organ, oscillo
scope, ostrich, otter, otterhound, overskirt, ox, oxcart, oxygen_mask, oystercatch
er, packet, paddle, paddlewheel, padlock, paintbrush, pajama, palace, panpipe, pap
er_towel, papillon, parachute, parallel_bars, park_bench, parking_meter, partridg
e, passenger_car, patas, patio, pay-phone, peacock, pedestal, pelican, pencil_box,
pencil_sharpener, perfume, photocopier, pick, pickelhaube, picket_fence, pickup, p
ier, piggy_bank, pill_bottle, pillow, pineapple, ping-pong_ball, pinwheel, pirate,
pitcher, pizza, plane, planetarium, plastic_bag, plate, plate_rack, platypus, plo
w, plunger, pole, polecat, police_van, pomegranate, poncho, pool_table, pop_bottl
e, porcupine, pot, potpie, potter's_wheel, power_drill, prairie_chicken, prayer_ru
g, pretzel, printer, prison, proboscis_monkey, projectile, projector, promontory,
ptarmigan, puck, puffer, pug, punching_bag, purse, quail, quill, quilt, racer, rac
ket, radiator, radio, radio_telescope, rain_barrel, ram, rapeseed, recreational_ve
hicle, red-backed_sandpiper, red-breasted_merganser, red_fox, red_wine, red_wolf,
redbone, redshank, reel, reflex_camera, refrigerator, remote_control, restaurant,
revolver, rhinoceros_beetle, rifle, ringlet, ringneck_snake, robin, rock_beauty, r
ock_crab, rock_python, rocking_chair, rotisserie, rubber_eraser, ruddy_turnstone,
ruffed_grouse, rugby_ball, rule, running_shoe, safe, safety_pin, saltshaker, sanda
l, sandbar, sarong, sax, scabbard, scale, schipperke, school_bus, schooner, scoreb
oard, scorpion, screen, screw, screwdriver, scuba_diver, sea_anemone, sea_cucumbe
r, sea_lion, sea_slug, sea_snake, sea_urchin, seashore, seat_belt, sewing_machine,
shield, shoe_shop, shoji, shopping_basket, shopping_cart, shovel, shower_cap, show
er_curtain, siamang, sidewinder, silky_terrier, ski, ski_mask, skunk, sleeping_ba
g, slide_rule, sliding_door, slot, sloth_bear, slug, snail, snorkel, snow_leopard,
snowmobile, snowplow, soap_dispenser, soccer_ball, sock, soft-coated_wheaten_terri
er, solar_dish, sombrero, sorrel, soup_bowl, space_bar, space_heater, space_shuttl
e, spaghetti_squash, spatula, speedboat, spider_monkey, spider_web, spindle, spiny
_lobster, spoonbill, sports_car, spotlight, spotted_salamander, squirrel_monkey, s
tage, standard_poodle, standard_schnauzer, starfish, steam_locomotive, steel_arch_
bridge, steel_drum, stethoscope, stingray, stinkhorn, stole, stone_wall, stopwatc
h, stove, strainer, strawberry, street_sign, streetcar, stretcher, studio_couch, s
tupa, sturgeon, submarine, suit, sulphur-crested_cockatoo, sulphur_butterfly, sund
ial, sunglass, sunglasses, sunscreen, suspension_bridge, swab, sweatshirt, swimmin
g_trunks, swing, switch, syringe, tabby, table_lamp, tailed_frog, tank, tape_playe
r, tarantula, teapot, teddy, television, tench, tennis_ball, terrapin, thatch, the
ater_curtain, thimble, three-toed_sloth, thresher, throne, thunder_snake, tick, ti
ger, tiger_beetle, tiger_cat, tiger_shark, tile_roof, timber_wolf, titi, toaster,
tobacco_shop, toilet_seat, toilet_tissue, torch, totem_pole, toucan, tow_truck, to
y_poodle, toy_terrier, toyshop, tractor, traffic_light, trailer_truck, tray, tree_
frog, trench_coat, triceratops, tricycle, trifle, trilobite, trimaran, tripod, tri
umphal_arch, trolleybus, trombone, tub, turnstile, tusker, typewriter_keyboard, um
brella, unicycle, upright, vacuum, valley, vase, vault, velvet, vending_machine, v
estment, viaduct, vine_snake, violin, vizsla, volcano, volleyball, vulture, waffle
_iron, walking_stick, wall_clock, wallaby, wallet, wardrobe, warplane, warthog, wa
shbasin, washer, water_bottle, water_buffalo, water_jug, water_ouzel, water_snake,
water_tower, weasel, web_site, weevil, whippet, whiptail, whiskey_jug, whistle, wh

```
ite_stork, white_wolf, wig, wild_boar, window_screen, window_shade, wine_bottle, w
ing, wire-haired_fox_terrier, wok, wolf_spider, wombat, wood_rabbit, wooden_spoon,
wool, worm_fence, wreck, yawl, yellow_lady's_slipper, yurt, zebra, zucchini
```

- Predict on the loaded image

In [10]:
```
# make a prediction on the image
preds = model.predict(xi)

# decode the results into a list of tuples (class, description, probability)
predsc = resnet.decode_predictions(preds, top=3)[0]

for i in predsc:
    print(i)
```

```
2023-01-23 16:00:35.761484: W tensorflow/core/platform/profile_utils/cpu_utils.cc:
128] Failed to get CPU frequency: 0 Hz
2023-01-23 16:00:35.934258: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
('n02504013', 'Indian_elephant', 0.84596217)
('n01871265', 'tusker', 0.15151736)
('n02504458', 'African_elephant', 0.0025194385)
```

Pre-trained Image Features

- The network is trained on ImageNet, which contains many classes and images.
- The learned features generalize well to other tasks
  - capture high-level discriminative information about objects

In [11]:
```
# load dessert dataset
import glob
import os

imglist = glob.glob(os.sep.join(['data','dessert', '*', '*.jpg']))
Xraw = []
Xim = zeros((len(imglist), 224, 224, 3))
Ylab = []
for i,img_path in enumerate(imglist):
    img = image.load_img(img_path, target_size=(224, 224))
    Xraw.append(img)
    x = image.img_to_array(img)
    x = expand_dims(x, axis=0)
    x = resnet.preprocess_input(x)
    Xim[i,:] = x
    Ylab.append(img_path.split(os.sep)[2])
print(Xim.shape)
print(len(Ylab))
```

```
(200, 224, 224, 3)
200
```

In [12]:
```
# convert class strings into integers
print("class labels (strings):", unique(Ylab))
le = preprocessing.LabelEncoder()
Y = le.fit_transform(Ylab)
print("Converted labels:")
print(Y)
```

```
class labels (strings): ['frozen_yogurt' 'ice_cream']
Converted labels:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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]
```

- Show a few examples of "ice cream" and "frozen yogurt"

In [13]:
```
inds = [0, 1, 2, 3, 4, -1, -2, -3, -4, -5]
tmp = [Xraw[i] for i in inds]
t =  [Ylab[i] for i in inds]
plt.figure(figsize=(9,4))
show_imgs(tmp, nc=5, titles=t)
```

Computing Image Features

- Use the pre-trained network as a feature extractor
  - remove the last layer (the classifier)
  - apply average pooling on the feature map
    - 7x7x2048 --> 1x2048

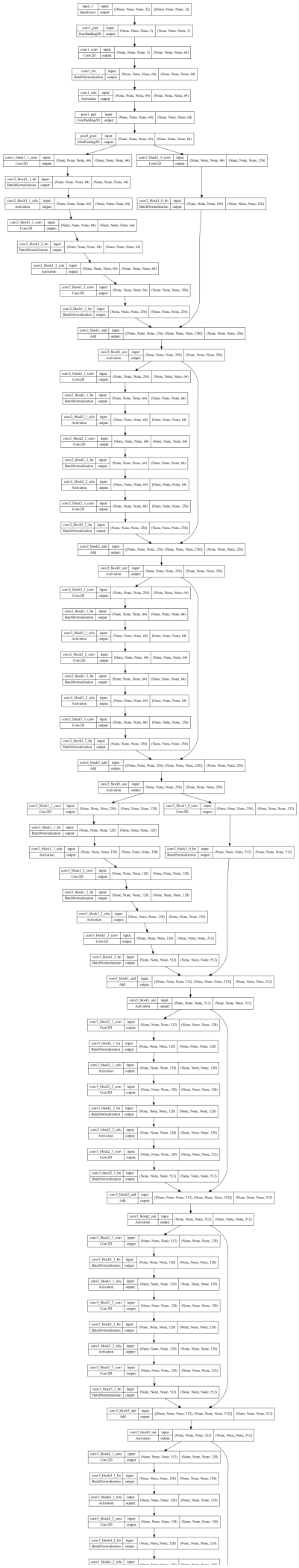
In [14]:
```
# create an instance of the model w/o the last layer
model_f = resnet.ResNet50(weights='imagenet',
                          include_top=False,  # remove the classification layer
                          pooling='avg')      # apply GlobalAveragePooling

# compute the features
Xf = model_f.predict(Xim)
print(Xf.shape)
```

```
2023-01-23 16:00:37.949452: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
(200, 2048)
```

In [15]:
```
tmp = tf.keras.utils.plot_model(model_f, to_file='tmp_model.png', show_shapes=True)
tmp.width = 400
tmp
```

Out[15]:



- Now use the features to train a binary classifier
- First setup the dataset

In [16]:
```
# convert class labels to binary indicators
Yb = keras.utils.to_categorical(Y)

# randomly split data into train and test set
( trainXf, testXf,        # features
  trainY, testY,          # class labels
  trainXim, testXim,      # pre-processed images
  trainYb, testYb ,       # binary class vector (for Keras)
  trainXraw, testXraw     # original images
) = \
  model_selection.train_test_split(Xf, Y, Xim, Yb, Xraw,
  train_size=0.8, test_size=0.2, random_state=4487)

trainX = trainXim.reshape((trainXim.shape[0], -1))
testX  = testXim.reshape((testXim.shape[0], -1))

print(trainXf.shape)
print(testXf.shape)
print(trainXim.shape)
print(trainX.shape)
```

```
(160, 2048)
(40, 2048)
(160, 224, 224, 3)
(160, 150528)
```

- Use the image features to train a standard SVM classifier

In [17]:
```
# setup the list of parameters to try
paramgrid = {'C': logspace(-3,3,13)}
print(paramgrid)

# setup cross-validation
svmcv = model_selection.GridSearchCV(svm.SVC(kernel='linear'), paramgrid, cv=5,
                                     n_jobs=-1, verbose=True)

# run cross-validation (train for each split)
svmcv.fit(trainXf, trainY);
```

```
{'C': array([1.00000000e-03, 3.16227766e-03, 1.00000000e-02, 3.16227766e-02,
       1.00000000e-01, 3.16227766e-01, 1.00000000e+00, 3.16227766e+00,
       1.00000000e+01, 3.16227766e+01, 1.00000000e+02, 3.16227766e+02,
       1.00000000e+03])}
Fitting 5 folds for each of 13 candidates, totalling 65 fits
```

- Predict with the classifier
- The accuracy is great!

In [18]:
```
# predict
predY = svmcv.predict(testXf)

acc = metrics.accuracy_score(testY, predY)
print("test accuracy = " + str(acc))
```

```
test accuracy = 0.95
```

- For comparison, also train an SVM on the raw images
- The accuracy is really bad (almost chance level)

In [19]:
```
svmcv_im = model_selection.GridSearchCV(svm.SVC(kernel='linear'), paramgrid, cv=5,
                                        n_jobs=-1, verbose=True)

# run cross-validation (train for each split)
```

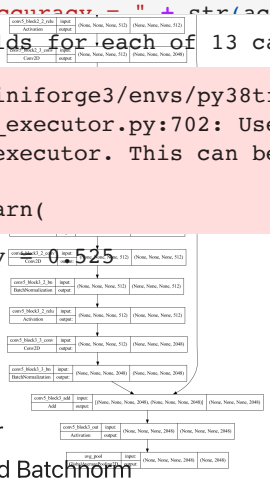
```
svmcv_im.fit(trainX, trainY);


# Directly use svmcv to make predictions
predY = svmcv_im.predict(testX)

acc = metrics.accuracy_score(testY, predY)
print("test accuracy = " + str(acc))
```

```
Fitting 5 folds for each of 13 candidates, totalling 65 fits
```

```
/Users/abc/miniforge3/envs/py38tfm28/lib/python3.8/site-packages/joblib/externals/
loky/process_executor.py:702: UserWarning: A worker stopped while some jobs were g
iven to the executor. This can be caused by a too short worker timeout or by a mem
ory leak.
  warnings.warn(
```

```
test accuracy = 0.525
```

Outline



- Going deeper
  - ReLU and Batchnorm
- Optimization methods
- Deep architectures and Image classification
- **Transfer learning**

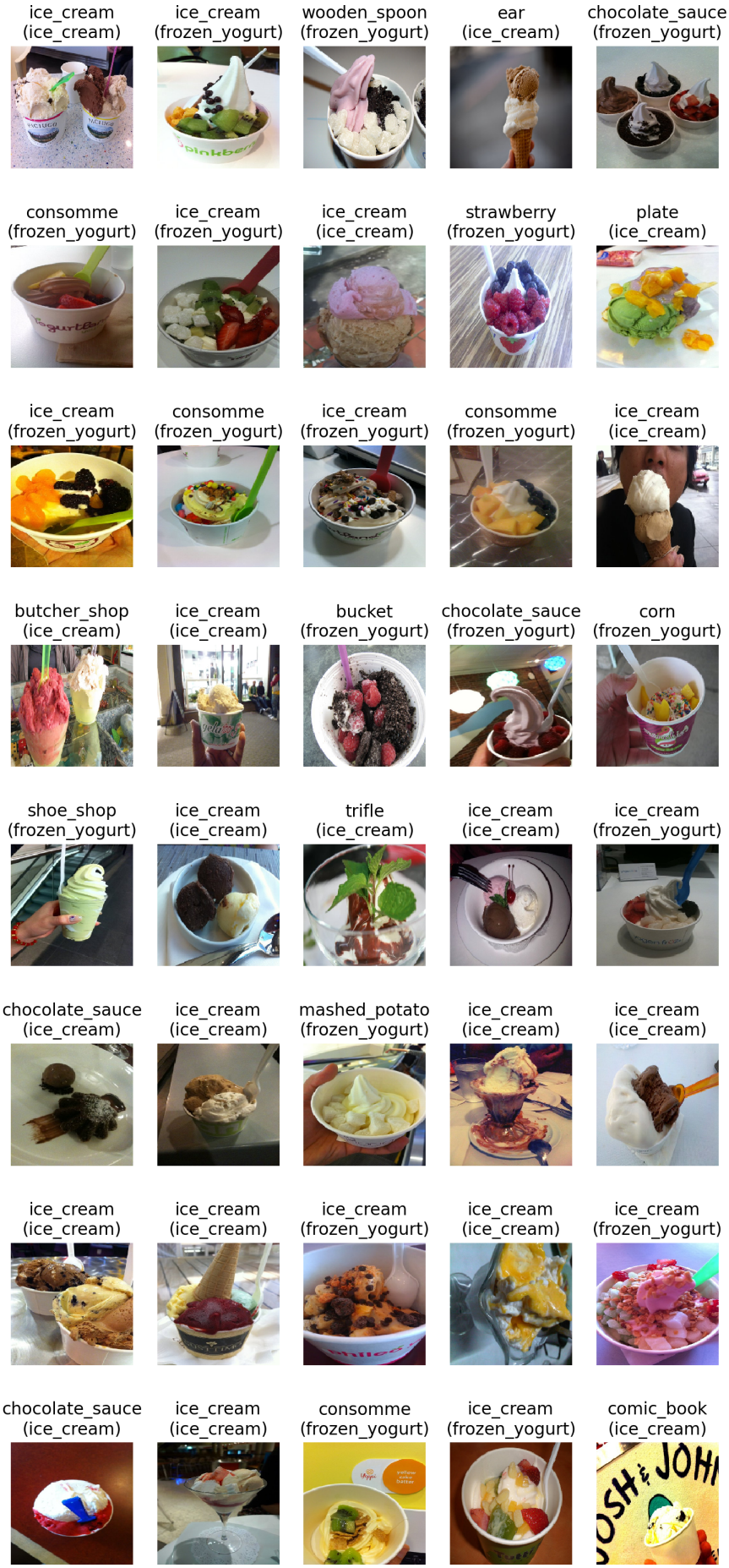
Transfer learning and fine-tuning

- The pre-trained network can also be "fine-tuned" for a new image classification task.
  - This is called transfer learning.
- We could:
  1. start from the pre-trained network, and train the whole network on the new classification task
  2. Rather than retrain the whole network
     - fix the lower layers that extract features (no need to train them since they are good features)
     - train the last few layers to perform the new task
- Does not need as much data, compared to the original ImageNet.

- First, let's see ResNet50 predicts on our images

In [20]:
```
# try the original ResNet50
testYscore = model.predict(testXim)

# get the class labels
predY = resnet.decode_predictions(testYscore, top=1)
testYcl = le.inverse_transform(testY)
```

In [22]:
```
# make a plot: predicted (true)
plt.figure(figsize=(9,20))
show_imgs(testXraw, nc=5, titles=titles)
```



- We will use the `Model` class
  - allows for more complex layer interactions than "Sequential"
  - first, connect layers one-by-one
    - instantiate the layer (with hyperparameters)

- then pass the previous layer to it

In [23]:
```
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)

# create the base pre-trained model with-out the classifier
# using global average pooling
base_model = resnet.ResNet50(weights='imagenet', include_top=False, pooling='avg')

# start with the output of the ResNet50 (1x1x2048)
x = base_model.output

# fully-connected layer (1 x32)
# (only two classes so don't need so many)
x = Dense(32, activation='relu')(x)

# finally, the softmax for the classifier (2 classes)
predictions = Dense(2, activation='softmax')(x)
```

- Second, create the Model
  - specify the input and output layers
  - the network is everything in between
- Fix the weights of the layers of ResNet so they will not change during training

In [24]:
```
# build the model for training
# - need to specify the input layer and the output layer
model_ft = Model(inputs=base_model.input, outputs=predictions)

# fix the layers of the ResNet50.
for layer in base_model.layers:
    layer.trainable = False

# compile the model - only the layers that we added will be trained
model_ft.compile(optimizer='rmsprop',
              loss='categorical_crossentropy', metrics=['accuracy'])
```

- Setup validation set for monitoring

In [25]:
```
# generate a fixed validation set using 10% of the training set
vtrainXim, validXim, vtrainYb, validYb = \
  model_selection.train_test_split(trainXim, trainYb,
  train_size=0.9, test_size=0.1, random_state=6487)

# validation data
validset  = (validXim, validYb)
```

- Setup the data augmentation generator

In [26]:
```
def add_gauss_noise(X, sigma2=0.1):  #0.05
    # add Gaussian noise with zero mean, and variance sigma2
    return X + random.normal(0, sigma2, X.shape)

# build the data augmenter
datagen = ImageDataGenerator(
    rotation_range=10,         # image rotation
    width_shift_range=0.1,     # image shifting
    height_shift_range=0.1,    # image shifting
    shear_range=0.1,           # shear transformation
    zoom_range=0.1,            # zooming
    horizontal_flip=True,
    preprocessing_function=add_gauss_noise,
)

# fit (required for some normalization augmentations)
datagen.fit(vtrainXim)
```

- Train the model

- only the last layers that we added are updated

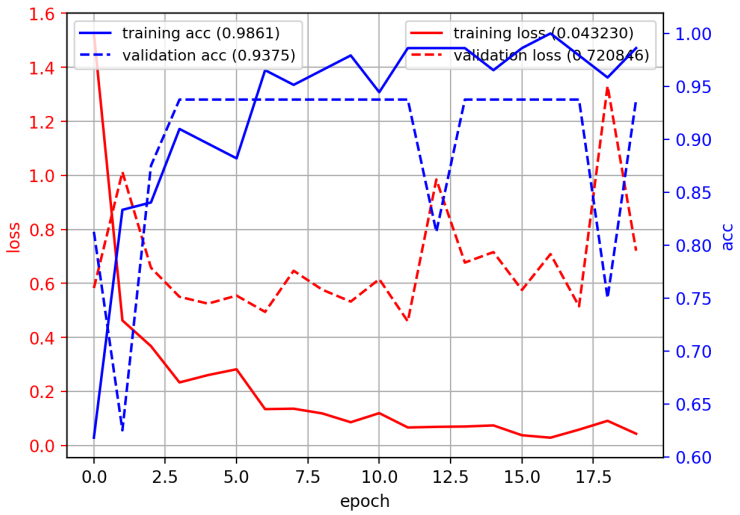
In [28]:
```
# train the model on the new data for a few epochs
bsize = 32
callbacks_list = []
history = model_ft.fit(
            datagen.flow(vtrainXim, vtrainYb, batch_size=bsize),  # data from generator
            steps_per_epoch=len(vtrainXim)/bsize,    # should be number of batches per epoch
            epochs=20,
            callbacks=callbacks_list,
            validation_data=validset, verbose=True)
```

```
Epoch 1/20
2023-01-23 16:01:05.336423: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
5/4 [==============================] - ETA: 0s - loss: 1.5256 - accuracy: 0.618
1
2023-01-23 16:01:07.029894: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
4/4 [==============================] - 3s 410ms/step - loss: 1.5256 - accuracy: 0.
6181 - val_loss: 0.5827 - val_accuracy: 0.8125
Epoch 2/20
4/4 [==============================] - 1s 264ms/step - loss: 0.4628 - accuracy: 0.
8333 - val_loss: 1.0124 - val_accuracy: 0.6250
Epoch 3/20
4/4 [==============================] - 1s 263ms/step - loss: 0.3675 - accuracy: 0.
8403 - val_loss: 0.6579 - val_accuracy: 0.8750
Epoch 4/20
4/4 [==============================] - 1s 267ms/step - loss: 0.2331 - accuracy: 0.
9097 - val_loss: 0.5502 - val_accuracy: 0.9375
Epoch 5/20
4/4 [==============================] - 1s 312ms/step - loss: 0.2604 - accuracy: 0.
8958 - val_loss: 0.5249 - val_accuracy: 0.9375
Epoch 6/20
4/4 [==============================] - 1s 265ms/step - loss: 0.2817 - accuracy: 0.
8819 - val_loss: 0.5547 - val_accuracy: 0.9375
Epoch 7/20
4/4 [==============================] - 1s 247ms/step - loss: 0.1342 - accuracy: 0.
9653 - val_loss: 0.4943 - val_accuracy: 0.9375
Epoch 8/20
4/4 [==============================] - 1s 250ms/step - loss: 0.1359 - accuracy: 0.
9514 - val_loss: 0.6466 - val_accuracy: 0.9375
Epoch 9/20
4/4 [==============================] - 1s 256ms/step - loss: 0.1184 - accuracy: 0.
9653 - val_loss: 0.5767 - val_accuracy: 0.9375
Epoch 10/20
4/4 [==============================] - 1s 246ms/step - loss: 0.0857 - accuracy: 0.
9792 - val_loss: 0.5325 - val_accuracy: 0.9375
Epoch 11/20
4/4 [==============================] - 1s 255ms/step - loss: 0.1194 - accuracy: 0.
9444 - val_loss: 0.6155 - val_accuracy: 0.9375
Epoch 12/20
4/4 [==============================] - 1s 256ms/step - loss: 0.0663 - accuracy: 0.
9861 - val_loss: 0.4592 - val_accuracy: 0.9375
Epoch 13/20
4/4 [==============================] - 1s 248ms/step - loss: 0.0686 - accuracy: 0.
9861 - val_loss: 0.9846 - val_accuracy: 0.8125
Epoch 14/20
4/4 [==============================] - 1s 255ms/step - loss: 0.0699 - accuracy: 0.
9861 - val_loss: 0.6770 - val_accuracy: 0.9375
Epoch 15/20
4/4 [==============================] - 1s 253ms/step - loss: 0.0739 - accuracy: 0.
9653 - val_loss: 0.7158 - val_accuracy: 0.9375
Epoch 16/20
4/4 [==============================] - 1s 246ms/step - loss: 0.0375 - accuracy: 0.
9861 - val_loss: 0.5756 - val_accuracy: 0.9375
Epoch 17/20
4/4 [==============================] - 1s 259ms/step - loss: 0.0285 - accuracy: 1.
0000 - val_loss: 0.7084 - val_accuracy: 0.9375
Epoch 18/20
4/4 [==============================] - 1s 258ms/step - loss: 0.0583 - accuracy: 0.
```

```
9792 - val_loss: 0.5151 - val_accuracy: 0.9375
Epoch 19/20
4/4 [==============================] - 1s 258ms/step - loss: 0.0910 - accuracy: 0.
9583 - val_loss: 1.3309 - val_accuracy: 0.7500
Epoch 20/20
4/4 [==============================] - 1s 256ms/step - loss: 0.0432 - accuracy: 0.
9861 - val_loss: 0.7208 - val_accuracy: 0.9375
```

- Training/validation curves

In [29]:
```
plot_history(history)
```



- Compute test accuracy

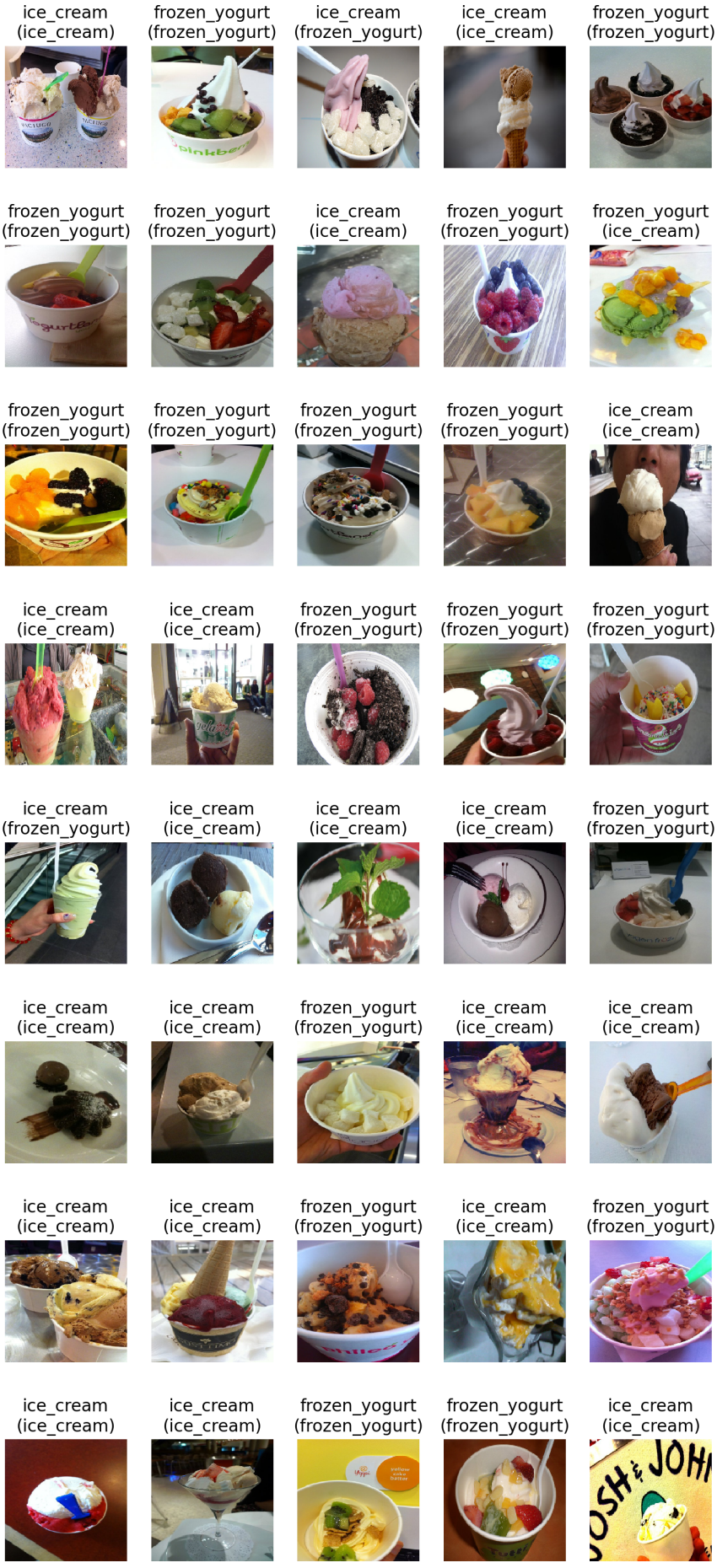
In [30]:
```
predY = argmax(model_ft.predict(testXim, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
2023-01-23 16:01:31.127798: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
test accuracy: 0.925
```

- Visualize the predictions

In [32]:
```
# make a plot: predicted (true)
plt.figure(figsize=(9,20))
show_imgs(testXraw, nc=5, titles=titles)
```



Using more features

- the last layers of ResNet are high-level semantic features, which could be complemented by low-level or mid-level features.

- extract features from multiple layers and concatenate them.

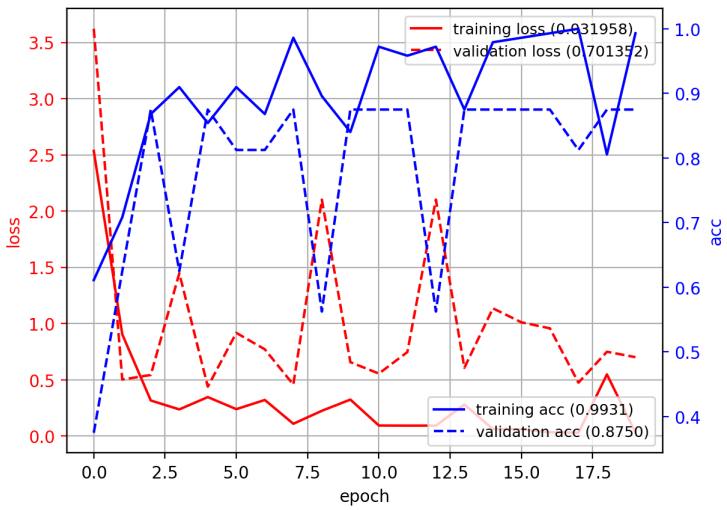
In [35]:
```
# create the base pre-trained model with-out the classifier
base_model = resnet.ResNet50(weights='imagenet', include_top=False, pooling='avg')

# start with the output of the ResNet50 (1x1x2048)
x = base_model.output

# get a middle layer output (end of one block), and apply GlobalAveragePooling
y = base_model.get_layer('conv3_block4_out').output
y = GlobalAveragePooling2D()(y)

# concatenate the high-level and mid-level features together
x = Concatenate()([x, y])
x = Dense(64, activation='relu')(x)   # fully-connected layer
predictions = Dense(2, activation='softmax')(x)  # softmax classifier (2 classes)

# build the model for training
# - need to specify the input layer and the output layer
model_ft = Model(inputs=base_model.input, outputs=predictions)

# fix the layers of the ResNet50.
for layer in base_model.layers:
    layer.trainable = False
```

- similar accuracy when using more features

In [37]:
```
plot_history(history)
predY = argmax(model_ft.predict(testXim, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
2023-01-23 16:02:15.431735: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
test accuracy: 0.925
```



Other tips

- If the dataset is too large to load into memory, can use `tf.keras.utils.image_dataset_from_directory` to build a dataset from directory.
  - will handle disk reading and caching of files.
- each pre-trained network has its own `preprocess_input` function.
  - this function can also be used as a layer if you want to embed preprocessing into the network.
  - the expected input (pixel range and image size) varies - check the documentation.

Summary

- Network "tricks" to go deeper:
  - ReLU activations - encourage sparse activations, reduce vanishing gradient problem.
  - Batch Normalization - reparameterizing the network to be more stable.
- Optimization "tricks":
  - SGD with decaying learning rate
    - analysis shows that noisy gradient actually helps to ignore the small local minima.
  - Adaptive learning rates: AdaGrad, RMSProp, Adam.

- Deep architectures:
  - advances of deep learning has been driven by the ImageNet competiton.
  - error rate decreases as the depth increases.
  - as depth increases, need to have a smart architecture design to make training more effective.
    - VGGnet - deep version of standard CNN
    - InceptionNet - auxiliary tasks
    - ResNet - residual learning, equivalent to an ensemble of models.
    - SENet - channel-wise attention
  - Vision Transformer & self-attention
    - more training data can trump the inductive bias in CNNs
    - (at the cost of computation and parameters)

- Transfer learning
  - feature extractors learned from ImageNet can be re-used for other tasks.
    - global pooled feataures for high-level tasks
    - mid-level and high-level feature maps for other tasks like localization and detection.
  - fine-tune a pre-trained network on our own specific task.
    - use the pre-trained network as a "backbone" network.
  - Pre-trained models: https://keras.io/api/applications/