Multi-class classification with Convolutional Neural Networks and Transfer Learning

on Canadian Institute For Advanced Research 10 dataset

Liana Isayan

Content

- Problem & data
- Data preparation
- Model building: model1
- Model building: model1
- Model building: model1
- Transfer learning: VGG16
- Transfer learning: VGG19
- Transfer learning: Inception v3
- Transfer learning: Resnet50 v2
- Visualizing predicted images
- References

Problem & Data

CIFAR-10 (**Canadian Institute For Advanced Research**) is a collection of images with 10 different classes representing **airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks**. CIFAR-10 is a set of images that can be used to teach a computer how to recognize objects.

The CIFAR-10 dataset consists of 60000 32x32x3 i.e. color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. You can learn more about this dataset from here - https://www.cs.toronto.edu/~kriz/cifar.html

Since the images in CIFAR-10 are low-resolution (32x32x3), this dataset can allow researchers to quickly try different algorithms to see what works.

A multi-class classification algorithm to predict 10 different classes of the CIFAR-10 dataset using Convolutional Neural Networks and Transfer Learning will be built here.

Here the data is stored in a 4-dimensional NumPy array. The first dimension 50000 is denoting the number of images, the second dimension 32 is denoting the number of pixels along the x-axis, the third dimension 32 is denoting the number of pixels along the y-axis and the fourth dimension 3 is the total number of channels in those images

```
x_train.shape
```

```
(50000, 32, 32, 3)
```

```
y_train[0]
```

```
array([6], dtype=uint8)
```

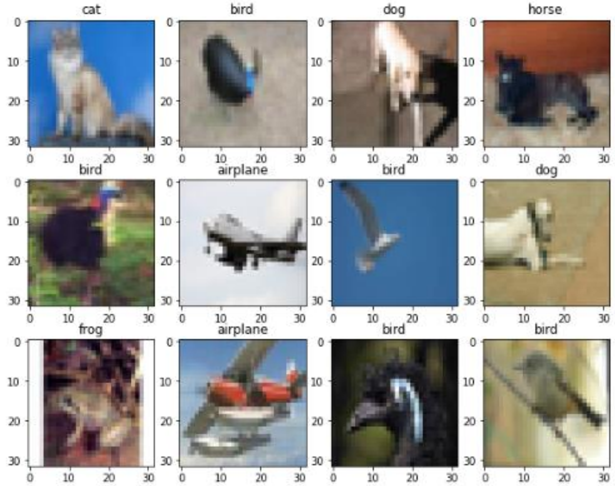
```
x_test.shape
```

```
(10000, 32, 32, 3)
```

```
x_train[0]
```

```
array([[[ 59,  62,  63],
        [ 43,  46,  45],
        [ 50,  48,  43],
        ...,
        [158, 132, 108],
        [152, 125, 102],
        [148, 124, 103]],

       [[ 16,  20,  20],
        [  0,   0,   0],
        [ 18,   8,   0],
        ...,
        [123,  88,  55],
        [119,  83,  50],
        [122,  87,  57]],

       [[ 25,  24,  21],
        [ 16,   7,   0],
        [ 49,  27,   8],
        ...,
        [118,  84,  50],
        [120,  84,  50],
        [109,  73,  42]],

       ...,
```

```
rows = 3
cols = 4
fig = plt.figure(figsize=(10, 8))
for i in range(cols):
    for j in range(rows):
        random_index = np.random.randint(0, len(y_train))
        ax = fig.add_subplot(rows, cols, i * rows + j + 1)
        ax.imshow(x_train[random_index, :])
        ax.set_title(cifar10_classes[y_train[random_index, 0]])
plt.show()
```



Data Preparation

normalize the feature inputs: As we know image pixel **values range from 0-255**, here we are simply **dividing all the pixel values by 255 to standardize all the images to have values between 0-1.**

```
# normalizing the image pixels
x_train_normalized = x_train/255
x_test_normalized = x_test/255
```

Since this is a **10 class classification problem**, the output layer should have **10 neurons** which will provide us with the probabilities of the input image belonging to each of those 10 classes. Therefore, we also need to create a ***one-hot encoded*** **representation for the target classes.**

```
# creating one-hot encoded representation of target labels
# we can do this by using this utility function - https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical

y_train_encoded = tf.keras.utils.to_categorical(y_train)
y_test_encoded = tf.keras.utils.to_categorical(y_test)
```

Model Building

fixing random states: **np.random.seed(42)** **import random** **random.seed(42)** **tf.random.set_seed(42)**

Model1: CNN model with Leaky Rectified Linear Unit (LeakyRelu)

```
model_1 = Sequential()

model_1.add(Conv2D(filters=16, kernel_size=(3, 3), padding="same", input_shape=(32, 32, 3)))

model_1.add(LeakyReLU(0.1))

model_1.add(Conv2D(filters=32, kernel_size=(3, 3), padding='same'))

model_1.add(LeakyReLU(0.1))

model_1.add(MaxPooling2D(pool_size=(2, 2)))

model_1.add(Flatten())

model_1.add(Dense(256))

model_1.add(LeakyReLU(0.1))

model_1.add(Dense(10, activation='softmax'))
```

```
# printing the model summary
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2097408
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 2,105,066
Trainable params: 2,105,066
Non-trainable params: 0
_________________________________________________________________
```

Model Building

Model1: CNN model with Leaky Rectified Linear Unit (LeakyRelu)

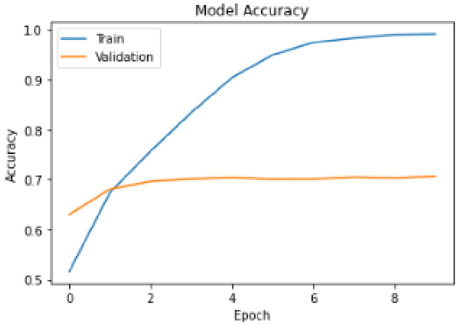
```
model_1.compile(
    loss='categorical_crossentropy', # as this is a multi-class classification problem
    optimizer=tf.keras.optimizers.Adamax(learning_rate=0.005), # here we are starting with 0.005 learning rate, default is 0.001
    metrics=['accuracy']
)
```

executed in 21ms, finished 19:05:16 2023-01-23

```
history_1 = model_1.fit(
            x_train_normalized, y_train_encoded,
            epochs=10,
            validation_split=0.1,
            shuffle=True,
            verbose=2
)
```

```
Epoch 1/10
1407/1407 - 39s - loss: 1.3549 - accuracy: 0.5161 - val_loss: 1.0402 - val_accuracy: 0.6304
Epoch 2/10
1407/1407 - 8s - loss: 0.9245 - accuracy: 0.6743 - val_loss: 0.9243 - val_accuracy: 0.6812
Epoch 3/10
1407/1407 - 8s - loss: 0.6921 - accuracy: 0.7571 - val_loss: 0.9210 - val_accuracy: 0.6968
Epoch 4/10
1407/1407 - 8s - loss: 0.4789 - accuracy: 0.8343 - val_loss: 0.9543 - val_accuracy: 0.7014
Epoch 5/10
1407/1407 - 8s - loss: 0.2823 - accuracy: 0.9041 - val_loss: 1.1085 - val_accuracy: 0.7042
Epoch 6/10
1407/1407 - 8s - loss: 0.1522 - accuracy: 0.9496 - val_loss: 1.3636 - val_accuracy: 0.7008
Epoch 7/10
1407/1407 - 8s - loss: 0.0819 - accuracy: 0.9746 - val_loss: 1.5808 - val_accuracy: 0.7012
Epoch 8/10
1407/1407 - 8s - loss: 0.0524 - accuracy: 0.9835 - val_loss: 1.8295 - val_accuracy: 0.7048
Epoch 9/10
1407/1407 - 8s - loss: 0.0321 - accuracy: 0.9903 - val_loss: 2.0032 - val_accuracy: 0.7032
Epoch 10/10
1407/1407 - 8s - loss: 0.0277 - accuracy: 0.9914 - val_loss: 2.0812 - val_accuracy: 0.7066
```

```
plt.plot(history_1.history['accuracy'])
plt.plot(history_1.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



Observations:

- We can see from the plots that the model has done poorly on the validation data. The model is highly overfitting the training data.
- The validation accuracy has become more or less constant after 2 epochs.

Let's try adding few dropout layers to the model structure to reduce overfitting and see if this improves the model or not.

Model Building

Clearing backend: **from tensorflow.keras import backend** **backend.clear_session()**

Model2: Model1 + adding few dropout layers

```
model_2 = Sequential()

model_2.add(Conv2D(filters=16, kernel_size=(3, 3), padding="same", input_shape=(32, 32, 3)))

model_2.add(LeakyReLU(0.1))

model_2.add(Dropout(0.2))

model_2.add(Conv2D(filters=32, kernel_size=(3, 3), padding='same'))

model_2.add(LeakyReLU(0.1))

model_2.add(Dropout(0.2))

model_2.add(MaxPooling2D(pool_size=(2, 2)))

model_2.add(Flatten())

model_2.add(Dense(256))

model_2.add(LeakyReLU(0.1))

model_2.add(Dropout(0.5))

model_2.add(Dense(10, activation='softmax'))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32, 32, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2097408
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 2,105,066
Trainable params: 2,105,066
Non-trainable params: 0
_________________________________________________________________
```

Model Building

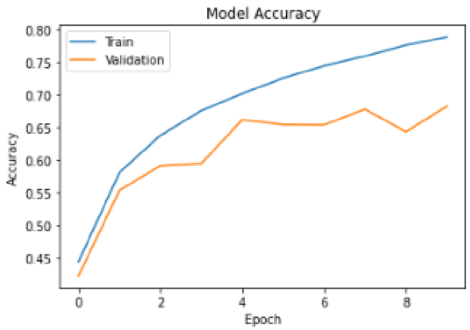
Model2: Model1 + adding few dropout layers

```
#compiling the model
model_2.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adamax(learning_rate=0.005),
    metrics=['accuracy']
)
```

```
#Fitting the model
history_2 = model_2.fit(
            x_train_normalized, y_train_encoded,
            epochs=10,
            validation_split=0.1,
            shuffle=True,
            verbose=2
)
```

```
Epoch 1/10
1407/1407 - 10s - loss: 1.5559 - accuracy: 0.4448 - val_loss: 1.7203 - val_accuracy: 0.4232
Epoch 2/10
1407/1407 - 9s - loss: 1.1794 - accuracy: 0.5814 - val_loss: 1.3611 - val_accuracy: 0.5540
Epoch 3/10
1407/1407 - 9s - loss: 1.0261 - accuracy: 0.6382 - val_loss: 1.2671 - val_accuracy: 0.5918
Epoch 4/10
1407/1407 - 9s - loss: 0.9262 - accuracy: 0.6762 - val_loss: 1.3171 - val_accuracy: 0.5950
Epoch 5/10
1407/1407 - 9s - loss: 0.8456 - accuracy: 0.7024 - val_loss: 1.0385 - val_accuracy: 0.6624
Epoch 6/10
1407/1407 - 9s - loss: 0.7817 - accuracy: 0.7260 - val_loss: 1.0738 - val_accuracy: 0.6548
Epoch 7/10
1407/1407 - 9s - loss: 0.7239 - accuracy: 0.7449 - val_loss: 1.1398 - val_accuracy: 0.6544
Epoch 8/10
1407/1407 - 9s - loss: 0.6775 - accuracy: 0.7598 - val_loss: 1.0062 - val_accuracy: 0.6784
Epoch 9/10
1407/1407 - 9s - loss: 0.6295 - accuracy: 0.7766 - val_loss: 1.1786 - val_accuracy: 0.6438
Epoch 10/10
1407/1407 - 9s - loss: 0.5974 - accuracy: 0.7890 - val_loss: 1.0203 - val_accuracy: 0.6830
```

```
plt.plot(history_2.history['accuracy'])
plt.plot(history_2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



Observations:

- The second model with dropout layers seems to have reduced the overfitting as compared to the previous model but still, the model is not performing well on the validation data.
- The validation accuracy has decreased slightly as compared to the previous model.

Let's now build another model with few more convolution layers, max-pooling layers, and dropout layers to reduce overfitting. Also, let's change the learning rate and the number of epochs and see if the model's performance improves.

Model Building

Model3: Model2 + with few more convolution, max-pooling, and dropout layers to reduce overfitting (also leaning rate and # of epochs changed)

```
model_3 = Sequential()

model_3.add(Conv2D(filters=16, kernel_size=(3, 3), padding="same", input_shape=(32, 32, 3)))

model_3.add(LeakyReLU(0.1))

model_3.add(Conv2D(filters=32, kernel_size=(3, 3), padding='same'))

model_3.add(LeakyReLU(0.1))

model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Dropout(0.25))

model_3.add(Conv2D(filters=32, kernel_size=(3, 3), padding='same'))

model_3.add(LeakyReLU(0.1))

model_3.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same'))

model_3.add(LeakyReLU(0.1))

model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Dropout(0.25))

model_3.add(Flatten())

model_3.add(Dense(256))

model_3.add(LeakyReLU(0.1))

model_3.add(Dropout(0.5))

model_3.add(Dense(10, activation='softmax'))
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 16)	448
leaky_re_lu (LeakyReLU)	(None, 32, 32, 16)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	4640
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 32)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	9248
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
leaky_re_lu_3 (LeakyReLU)	(None, 16, 16, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 256)	1048832
leaky_re_lu_4 (LeakyReLU)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

Total params: 1,084,234
Trainable params: 1,084,234
Non-trainable params: 0

Model Building

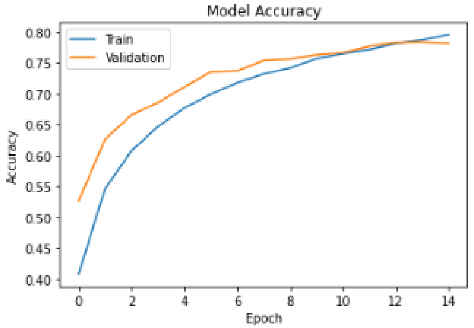
Model3: Model2 + with few more convolution, max-pooling, and dropout layers to reduce overfitting

```
model_3.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adamax(learning_rate=0.001),
    metrics=['accuracy']
)
```

```
history_3 = model_3.fit(
            x_train_normalized, y_train_encoded,
            epochs=15,
            validation_split=0.1,
            shuffle=True,
            verbose=2
)
```

```
Epoch 1/15
1407/1407 - 13s - loss: 1.6299 - accuracy: 0.4078 - val_loss: 1.3145 - val_accuracy: 0.5258
Epoch 2/15
1407/1407 - 12s - loss: 1.2723 - accuracy: 0.5454 - val_loss: 1.0702 - val_accuracy: 0.6256
Epoch 3/15
1407/1407 - 12s - loss: 1.1059 - accuracy: 0.6071 - val_loss: 0.9577 - val_accuracy: 0.6652
Epoch 4/15
1407/1407 - 12s - loss: 1.0001 - accuracy: 0.6464 - val_loss: 0.8973 - val_accuracy: 0.6852
Epoch 5/15
1407/1407 - 12s - loss: 0.9225 - accuracy: 0.6764 - val_loss: 0.8446 - val_accuracy: 0.7104
Epoch 6/15
1407/1407 - 12s - loss: 0.8583 - accuracy: 0.6988 - val_loss: 0.7723 - val_accuracy: 0.7348
Epoch 7/15
1407/1407 - 12s - loss: 0.8074 - accuracy: 0.7174 - val_loss: 0.7654 - val_accuracy: 0.7366
Epoch 8/15
1407/1407 - 12s - loss: 0.7677 - accuracy: 0.7314 - val_loss: 0.7182 - val_accuracy: 0.7532
Epoch 9/15
1407/1407 - 12s - loss: 0.7340 - accuracy: 0.7414 - val_loss: 0.7174 - val_accuracy: 0.7556
Epoch 10/15
1407/1407 - 12s - loss: 0.6937 - accuracy: 0.7561 - val_loss: 0.6883 - val_accuracy: 0.7622
Epoch 11/15
1407/1407 - 12s - loss: 0.6697 - accuracy: 0.7641 - val_loss: 0.6781 - val_accuracy: 0.7660
Epoch 12/15
1407/1407 - 12s - loss: 0.6452 - accuracy: 0.7712 - val_loss: 0.6446 - val_accuracy: 0.7766
Epoch 13/15
1407/1407 - 12s - loss: 0.6180 - accuracy: 0.7811 - val_loss: 0.6456 - val_accuracy: 0.7824
Epoch 14/15
1407/1407 - 12s - loss: 0.6008 - accuracy: 0.7865 - val_loss: 0.6359 - val_accuracy: 0.7832
Epoch 15/15
1407/1407 - 12s - loss: 0.5811 - accuracy: 0.7950 - val_loss: 0.6282 - val_accuracy: 0.7812
```

```
plt.plot(history_3.history['accuracy'])
plt.plot(history_3.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



There are a **few possible reasons** for the accuracy issue:

- The **size of the validation set is not big enough**
- We may have **imbalanced data in the validation set**
- **High regularization** (while evaluating the model, the model doesn't use regularization and hence there's no noise, which is why the validation accuracy doesn't decrease).
- **To overcome this, we can try reducing the regularization or increase the size of the validation set**

- The third iteration of this model seems very promising now.
- **The validation accuracy has improved substantially** and the problem of **overfitting has been reduced** completely. We can say that the **model is giving a generalized performance.**
- The above plot shows that **the validation accuracy is higher than the training accuracy**.

Transfer Learning

VGG16 as the pre-trained model. You can read about it here

We can try out some more iterations and tune some of the hyperparameters to further improve the model but hyperparameter tuning is exhaustive and can take a long time to find the right set of values for each hyperparameter.

Transfer learning is a popular deep learning technique that reuses a pre-trained model on a new problem. It can train deep neural networks with comparatively little data. This is very useful in the data science field since most real-world problems typically do not have millions of labeled data points to train complex models.

```
#Importing necessary libraries
from tensorflow.keras import Model
from tensorflow.keras.applications.vgg16 import VGG16
```

```
vgg_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(32, 32, 3), pooling='max')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kerne
ls_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
```

- The **VGG16 model was originally trained on images of size 224 x 224**. The **TensorFlow application allows the minimum image size of 32x32** which is luckily the same as the image size in the CIFAR-10 dataset.
- By specifying the **argument include_top=False argument**, we load a **network that doesn't include the classification layers at the top** i.e. **we will use the VGG16 model only for feature extraction.**

TensorFlow

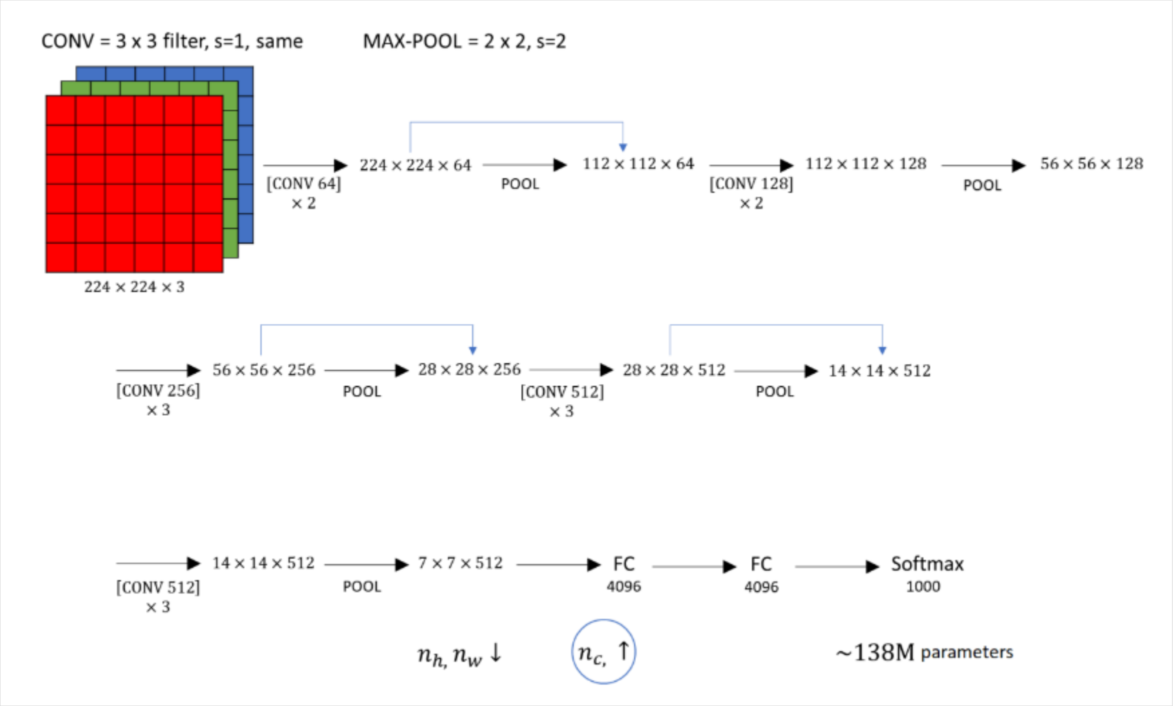
Filter

- applications
  - Overview
  - MobileNetV3Large
  - MobileNetV3Small
  - convnext
  - densenet
  - efficientnet
  - efficientnet_v2
  - imagenet_utils
  - inception_resnet_v2
  - inception_v3
  - mobilenet
  - mobilenet_v2
  - mobilenet_v3
  - nasnet
  - regnet
  - resnet
  - resnet50
  - resnet_rs
  - resnet_v2
  - vgg16
  - vgg19
  - xception

Transfer Learning

VGG16 as the pre-trained model. You can read about it [here]

- The VGG16 model has more than 14.7M trainable parameters.
- Also, we can take any layer's output from the VGG16 model as the input of our new model. I will take the output of the 3rd block as the input of new model.




```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
global_max_pooling2d (Global (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

```
transfer_layer = vgg_model.get_layer('block3_pool')
```

```
vgg_model.trainable=False
```

Transfer Learning

VGG16 as the pre-trained model. You can read about it here

2

```
#Compiling the model
model_4.compile(loss='categorical_crossentropy',
             optimizer=tf.keras.optimizers.Adamax(learning_rate = 0.0005),
             metrics=['accuracy'])
```

1

```
# Add classification layers on top of it

x = Flatten()(transfer_layer.output) #Flatten the output from the 3rd block of the VGG16 model
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = BatchNormalization()(x)
pred = Dense(10, activation='softmax')(x)

model_4 = Model(vgg_model.input, pred) #Initializing the model
```

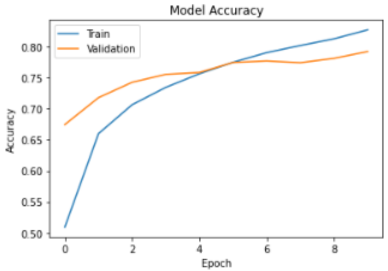
3

```
#Fitting the model
history_4 = model_4.fit(
            x_train_normalized, y_train_encoded,
            epochs=10,
            batch_size=250,
            validation_split=0.1,
            verbose=2
)
```

```
Epoch 1/10
180/180 - 8s - loss: 1.4544 - accuracy: 0.5092 - val_loss: 0.9604 - val_accuracy: 0.6746
Epoch 2/10
180/180 - 6s - loss: 1.0105 - accuracy: 0.6599 - val_loss: 0.8384 - val_accuracy: 0.7180
Epoch 3/10
180/180 - 6s - loss: 0.8663 - accuracy: 0.7066 - val_loss: 0.7626 - val_accuracy: 0.7428
Epoch 4/10
180/180 - 6s - loss: 0.7809 - accuracy: 0.7346 - val_loss: 0.7198 - val_accuracy: 0.7556
Epoch 5/10
180/180 - 6s - loss: 0.7137 - accuracy: 0.7566 - val_loss: 0.7008 - val_accuracy: 0.7586
Epoch 6/10
180/180 - 6s - loss: 0.6605 - accuracy: 0.7751 - val_loss: 0.6610 - val_accuracy: 0.7748
Epoch 7/10
180/180 - 6s - loss: 0.6162 - accuracy: 0.7906 - val_loss: 0.6498 - val_accuracy: 0.7772
Epoch 8/10
180/180 - 6s - loss: 0.5812 - accuracy: 0.8021 - val_loss: 0.6539 - val_accuracy: 0.7744
Epoch 9/10
180/180 - 6s - loss: 0.5405 - accuracy: 0.8129 - val_loss: 0.6313 - val_accuracy: 0.7814
Epoch 10/10
180/180 - 6s - loss: 0.5042 - accuracy: 0.8274 - val_loss: 0.6230 - val_accuracy: 0.7924
```

4

```
plt.plot(history_4.history['accuracy'])
plt.plot(history_4.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



- The model training accuracy is slightly higher than the validation accuracy.
- The validation accuracy has improved as compared to the previous model.
- We have been able to achieve the best validation accuracy so far without actually training any of the convolutional layers.

Transfer Learning

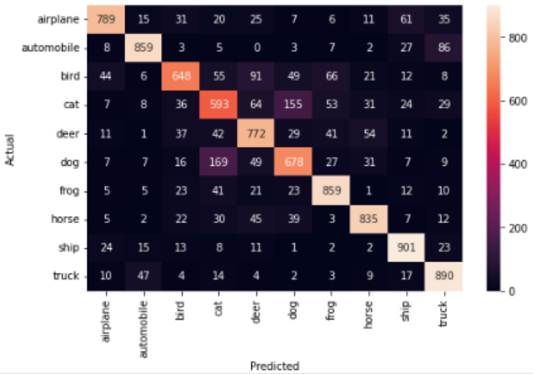
VGG16 as the pre-trained model. You can read about it [here]

Making predictions

```
#Making predictions on the test data
y_pred_test = model_4.predict(x_test_normalized)

#Converting probabilities to class labels
y_pred_test_classes = np.argmax(y_pred_test, axis=1)

#Calculating the probability of the predicted class
y_pred_test_max_probas = np.max(y_pred_test, axis=1)
```

```
#importing required functions
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

#Printing the classification report
print(classification_report(y_test, y_pred_test_classes))

#Plotting the heatmap using confusion matrix
cm = confusion_matrix(y_test, y_pred_test_classes)
plt.figure(figsize=(8,5))
sns.heatmap(cm, annot=True,  fmt='.0f', xticklabels=cifar10_classes, yticklabels=cifar10_classes)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

	precision	recall	f1-score	support
0	0.87	0.79	0.83	1000
1	0.89	0.86	0.87	1000
2	0.78	0.65	0.71	1000
3	0.61	0.59	0.60	1000
4	0.71	0.77	0.74	1000
5	0.69	0.68	0.68	1000
6	0.81	0.86	0.83	1000
7	0.84	0.83	0.84	1000
8	0.84	0.90	0.87	1000
9	0.81	0.89	0.85	1000
accuracy			0.78	10000
macro avg	0.78	0.78	0.78	10000
weighted avg	0.78	0.78	0.78	10000



- The model is giving about 78% accuracy on the test data which is comparable to the accuracy of the validation data. This implies that the model is giving a generalized performance.
- The recall has a high range which implies the model is good at identifying some objects while poor at some others. Eg, the model is able to identify more than 90% of ships but only ~65% dogs.
- The model is majorly confused between cat and dogs. This implies that the model might be focused on features related to shapes and sizes but not deep features of objects.
- Consequently, precision also has a high range with 'cat' class having the least precision.
- The highest precision is for 'horse' which implies that the model is able to distinguish horses from other objects.

Transfer Learning

VGG19 as the pre-trained model. You can read about it here

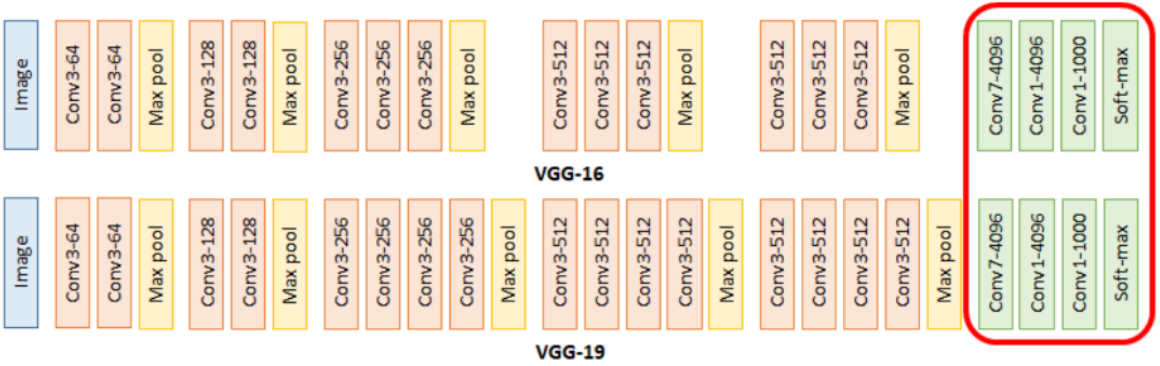
```
#Importing necessary libraries
from tensorflow.keras import Model
from tensorflow.keras.applications.vgg19 import VGG19
```

executed in 711ms, finished 00:10:27 2023-01-24

```
vgg_model_2 = VGG19(weights='imagenet',
                    include_top=False,
                    input_shape=(32, 32, 3), pooling='max')
```

executed in 27.1s, finished 00:16:01 2023-01-24

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 23s 0us/step
```




```
transfer_layer = vgg_model_2.get_layer('block5_pool')
```

```
vgg_model_2.trainable=False
```

```
x = Flatten()(transfer_layer.output) #Flatten the output from the 3rd block of the VGG16 model
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = BatchNormalization()(x)
pred = Dense(10, activation='softmax')(x)

model_5 = Model(vgg_model_2.input, pred) #Initializing the model
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv4 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv4 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv4 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
global_max_pooling2d_1 (GlobalMaxPooling2D)	(None, 512)	0

Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0

Transfer Learning

VGG19 as the pre-trained model. You can read about it here

```
#Compiling the model
model_5.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adamax(learning_rate = 0.0005),
              metrics=['accuracy'])
```

executed in 20ms, finished 00:21:42 2023-01-24

```
#Fitting the model
history_5 = model_5.fit(
            x_train_normalized, y_train_encoded,
            epochs=10,
            batch_size=250,
            validation_split=0.1,
            verbose=2
)
```

executed in 44m 23s, finished 01:10:07 2023-01-24

```
Epoch 1/10
180/180 - 235s - loss: 1.9756 - accuracy: 0.3174
Epoch 2/10
180/180 - 257s - loss: 1.6035 - accuracy: 0.4391
Epoch 3/10
180/180 - 315s - loss: 1.4765 - accuracy: 0.4806
Epoch 4/10
180/180 - 303s - loss: 1.4101 - accuracy: 0.5059
Epoch 5/10
180/180 - 300s - loss: 1.3635 - accuracy: 0.5223
Epoch 6/10
180/180 - 316s - loss: 1.3254 - accuracy: 0.5383
Epoch 7/10
180/180 - 291s - loss: 1.2937 - accuracy: 0.5512
Epoch 8/10
180/180 - 204s - loss: 1.2692 - accuracy: 0.5576
Epoch 9/10
180/180 - 212s - loss: 1.2474 - accuracy: 0.5631
Epoch 10/10
180/180 - 229s - loss: 1.2306 - accuracy: 0.5701
```

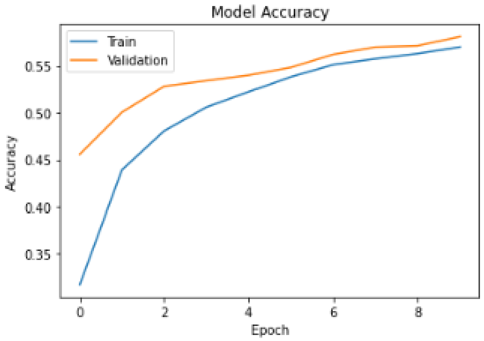
```
#Fitting the model
history_5 = model_5.fit(
            x_train_normalized, y_train_encoded,
            epochs=20,
            batch_size=250,
            validation_split=0.1,
            verbose=2
)
```

execution queued 01:52:49 2023-01-24

```
Epoch 1/20
180/180 - 289s - loss: 1.2420 - accuracy: 0.5654 - val_loss: 1.1986 - val_accuracy: 0.5714 - 289s/epoch - 2s/step
Epoch 2/20
180/180 - 376s - loss: 1.1996 - accuracy: 0.5811 - val_loss: 1.2029 - val_accuracy: 0.5684 - 376s/epoch - 2s/step
Epoch 3/20
180/180 - 312s - loss: 1.1696 - accuracy: 0.5896 - val_loss: 1.1699 - val_accuracy: 0.5846 - 312s/epoch - 2s/step
Epoch 4/20
180/180 - 423s - loss: 1.1467 - accuracy: 0.5984 - val_loss: 1.1884 - val_accuracy: 0.5818 - 423s/epoch - 2s/step
Epoch 5/20
180/180 - 756s - loss: 1.1295 - accuracy: 0.6050 - val_loss: 1.1928 - val_accuracy: 0.5750 - 756s/epoch - 4s/step
Epoch 6/20
180/180 - 743s - loss: 1.1080 - accuracy: 0.6134 - val_loss: 1.1686 - val_accuracy: 0.5880 - 743s/epoch - 4s/step
Epoch 7/20
180/180 - 741s - loss: 1.0870 - accuracy: 0.6209 - val_loss: 1.1432 - val_accuracy: 0.5906 - 741s/epoch - 4s/step
Epoch 8/20
180/180 - 734s - loss: 1.0738 - accuracy: 0.6239 - val_loss: 1.1492 - val_accuracy: 0.5944 - 734s/epoch - 4s/step
Epoch 9/20
180/180 - 710s - loss: 1.0603 - accuracy: 0.6286 - val_loss: 1.1316 - val_accuracy: 0.6024 - 710s/epoch - 4s/step
Epoch 10/20
180/180 - 709s - loss: 1.0458 - accuracy: 0.6324 - val_loss: 1.1337 - val_accuracy: 0.5968 - 709s/epoch - 4s/step
Epoch 11/20
180/180 - 716s - loss: 1.0351 - accuracy: 0.6373 - val_loss: 1.1269 - val_accuracy: 0.6010 - 716s/epoch - 4s/step
Epoch 12/20
180/180 - 715s - loss: 1.0163 - accuracy: 0.6428 - val_loss: 1.1286 - val_accuracy: 0.6058 - 715s/epoch - 4s/step
Epoch 13/20
180/180 - 715s - loss: 1.0062 - accuracy: 0.6466 - val_loss: 1.1338 - val_accuracy: 0.6026 - 715s/epoch - 4s/step
Epoch 14/20
```

```
plt.plot(history_5.history['accuracy'])
plt.plot(history_5.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

executed in 200ms, finished 01:10:07 2023-01-24

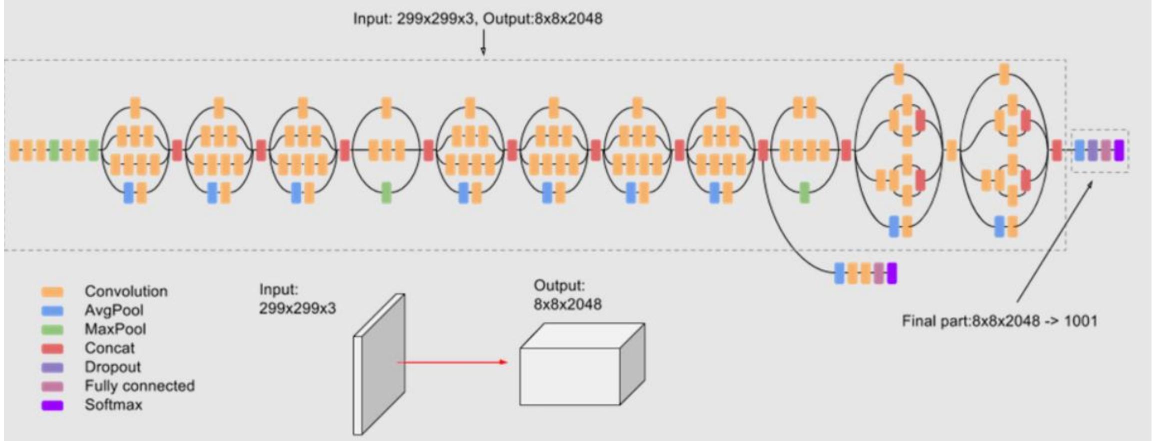


- The model validation accuracy is slightly higher than the training accuracy.
- The validation accuracy has surprisingly dropped as compared to the previous model.

Transfer Learning

Inception v3 as the pre-trained model. You can read about it here

- The main difference between Inception v1 and v3 is that Inception v3 uses more advanced techniques such as batch normalization and RMSprop optimizer which were not present in Inception V1. Additionally, Inception v3 is deeper and has more convolutional layers than v1, which allows it to learn more complex features from the input images. Another difference is that Inception v3 uses fewer parameters than v1, which makes it more computationally efficient.
- Inception v3 also includes a number of improvements to the Inception architecture such as the use of factorization and dimension reduction, which allows it to learn more efficiently from the input data and improve its accuracy.



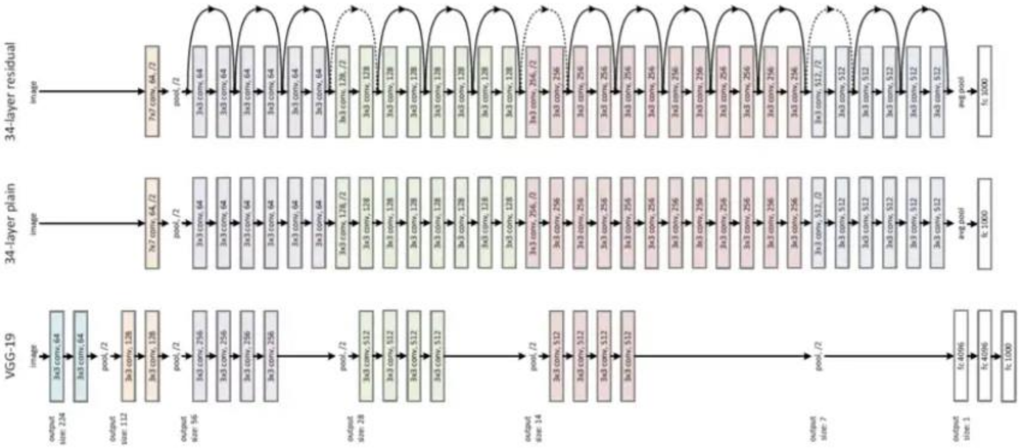

```
inception_model = InceptionV3(weights='imagenet',
                              include_top=False,
                              input_shape=(32, 32, 3), pooling='max')
```

executed in 35ms, finished 01:17:58 2023-01-24

```
---------------------------------------------------------------------------
ValueError: Input size must be at least 75x75; Received: input_shape=(32, 32, 3)
```

Transfer Learning

ResNet50V2 as the pre-trained model. You can read about it here




```
transfer_layer = resnet_model.get_layer('max_pool')
```

```
x = Flatten()(transfer_layer.output) #Flatten the output from the 3rd block of the VGG16 model
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = BatchNormalization()(x)
pred = Dense(10, activation='softmax')(x)

model_5 = Model(vgg_model_2.input, pred) #Initializing the model
```

```
resnet_model.summary()
```

executed in 357ms, finished 01:22:21 2023-01-24

```
Model: "resnet50v2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 32, 32, 3)]  0           []

 conv1_pad (ZeroPadding2D)      (None, 38, 38, 3)    0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 16, 16, 64)   9472        ['conv1_pad[0][0]']

 pool1_pad (ZeroPadding2D)      (None, 18, 18, 64)   0           ['conv1_conv[0][0]']

 pool1_pool (MaxPooling2D)      (None, 8, 8, 64)     0           ['pool1_pad[0][0]']

 conv2_block1_preact_bn (BatchN  (None, 8, 8, 64)    256         ['pool1_pool[0][0]']
 ormalization)

 conv2_block1_preact_relu (Acti  (None, 8, 8, 64)    0           ['conv2_block1_preact_bn[0][0]']
 vation)

 conv2_block1_1_conv (Conv2D)   (None, 8, 8, 64)     4096        ['conv2_block1_preact_relu[0][0]'
                                                                  ]

 conv2_block1_1_bn (BatchNormal  (None, 8, 8, 64)    256         ['conv2_block1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 8, 8, 64)    0           ['conv2_block1_1_bn[0][0]']
 n)

 conv2_block1_2_pad (ZeroPaddin  (None, 10, 10, 64)  0           ['conv2_block1_1_relu[0][0]']
 g2D)

 conv2_block1_2_conv (Conv2D)   (None, 8, 8, 64)     36864       ['conv2_block1_2_pad[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 8, 8, 64)    256         ['conv2_block1_2_conv[0][0]']
 ization)

 conv2_block1_2_relu (Activatio  (None, 8, 8, 64)    0           ['conv2_block1_2_bn[0][0]']
 n)

 post_bn (BatchNormalization)   (None, 1, 1, 2048)   8192        ['conv5_block3_out[0][0]']

 post_relu (Activation)         (None, 1, 1, 2048)   0           ['post_bn[0][0]']

 max_pool (GlobalMaxPooling2D)  (None, 2048)         0           ['post_relu[0][0]']

==================================================================================================
Total params: 23,564,800
Trainable params: 23,519,360
Non-trainable params: 45,440
__________________________________________________________________________________________________
```

Transfer Learning

ResNet50V2 as the pre-trained model. You can read about it here

```
#Compiling the model
model_7.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adamax(learning_rate = 0.0005),
              metrics=['accuracy'])
```

executed in 28ms, finished 01:26:14 2023-01-24

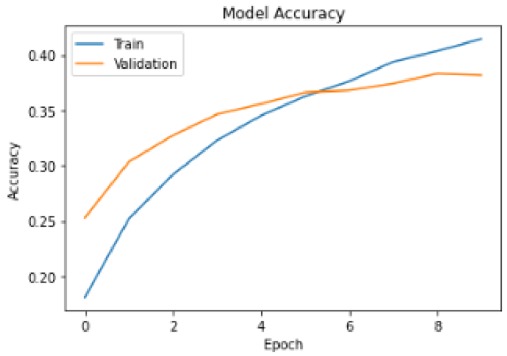
```
#Fitting the model
history_7 = model_7.fit(
            x_train_normalized, y_train_encoded,
            epochs=10,
            batch_size=250,
            validation_split=0.1,
            verbose=2
)
```

executed in 11m 54s, finished 01:38:13 2023-01-24

```
Epoch 1/10
180/180 - 76s - loss: 2.3627 - accuracy: 0.1811 - val_loss: 2.1825 - val_accuracy: 0.2530 - 76s/epoch - 423ms/step
Epoch 2/10
180/180 - 73s - loss: 2.1272 - accuracy: 0.2522 - val_loss: 2.0403 - val_accuracy: 0.3036 - 73s/epoch - 406ms/step
Epoch 3/10
180/180 - 69s - loss: 2.0130 - accuracy: 0.2921 - val_loss: 1.9331 - val_accuracy: 0.3274 - 69s/epoch - 382ms/step
Epoch 4/10
180/180 - 67s - loss: 1.9261 - accuracy: 0.3228 - val_loss: 1.8709 - val_accuracy: 0.3464 - 67s/epoch - 374ms/step
Epoch 5/10
180/180 - 67s - loss: 1.8594 - accuracy: 0.3452 - val_loss: 1.8317 - val_accuracy: 0.3558 - 67s/epoch - 371ms/step
Epoch 6/10
180/180 - 68s - loss: 1.8064 - accuracy: 0.3627 - val_loss: 1.8034 - val_accuracy: 0.3662 - 68s/epoch - 378ms/step
Epoch 7/10
180/180 - 72s - loss: 1.7604 - accuracy: 0.3762 - val_loss: 1.7854 - val_accuracy: 0.3680 - 72s/epoch - 402ms/step
Epoch 8/10
180/180 - 70s - loss: 1.7216 - accuracy: 0.3938 - val_loss: 1.7734 - val_accuracy: 0.3742 - 70s/epoch - 391ms/step
Epoch 9/10
180/180 - 74s - loss: 1.6904 - accuracy: 0.4036 - val_loss: 1.7641 - val_accuracy: 0.3832 - 74s/epoch - 413ms/step
Epoch 10/10
180/180 - 75s - loss: 1.6574 - accuracy: 0.4145 - val_loss: 1.7595 - val_accuracy: 0.3818 - 75s/epoch - 418ms/step
```

```
plt.plot(history_7.history['accuracy'])
plt.plot(history_7.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

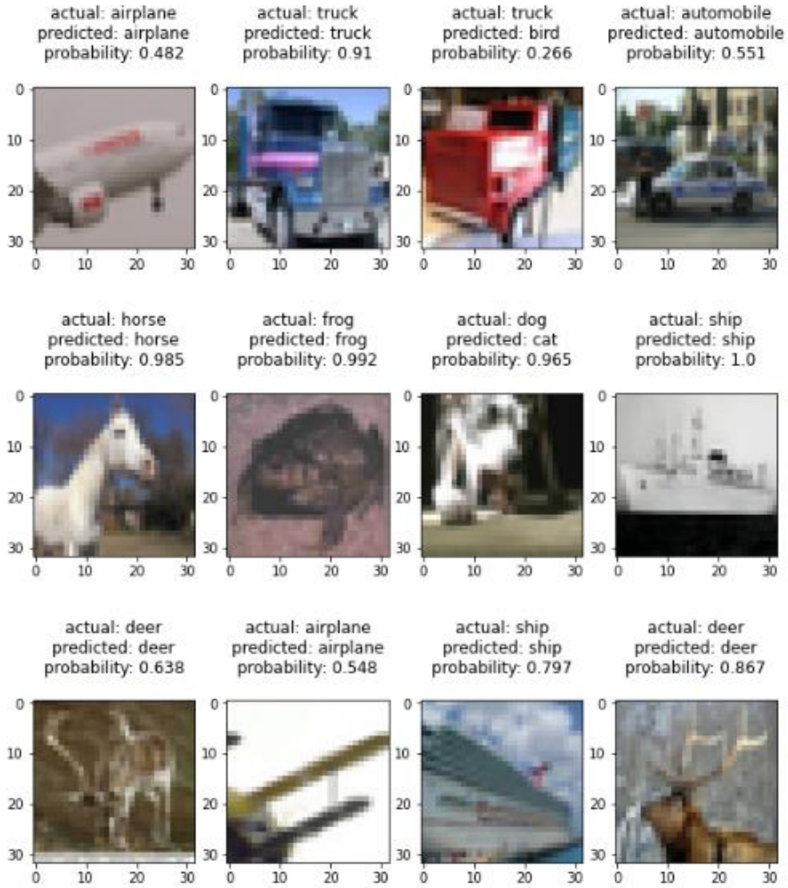
executed in 134ms, finished 01:38:13 2023-01-24



- The model training accuracy is slightly higher than the validation accuracy.
- The validation accuracy has improved as compared to the previous model.
- We have been able to achieve the best validation accuracy so far without actually training any of the convolutional layers.

Visualizing predicted images

```
rows = 3
cols = 4
fig = plt.figure(figsize=(10, 12))
for i in range(cols):
    for j in range(rows):
        random_index = np.random.randint(0, len(y_test))
        ax = fig.add_subplot(rows, cols, i * rows + j + 1)
        ax.imshow(x_test[random_index, :])
        pred_label = cifar10_classes[y_pred_test_classes[random_index]]
        pred_proba = y_pred_test_max_probas[random_index]
        true_label = cifar10_classes[y_test[random_index, 0]]
        ax.set_title("actual: {}\npredicted: {}\nprobability: {:.3}\n".format(
               true_label, pred_label, pred_proba
        ))
plt.show()
```



References

- Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna (2016), Rethinking the Inception Architecture for Computer Vision, *CVPR*
- G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken,
- and C. I. Sánchez (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88.
- Vazgen Mikayelyan(2023), *Deep Learning*, Lecture Notes
- https://www.tensorflow.org/datasets/catalog/cifar10
- https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg19/VGG19
- https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/VGG16
- https://www.tensorflow.org/api_docs/python/tf/keras/applications/inceptionv3/InceptionV3
- https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet_v2/ResNet50V2