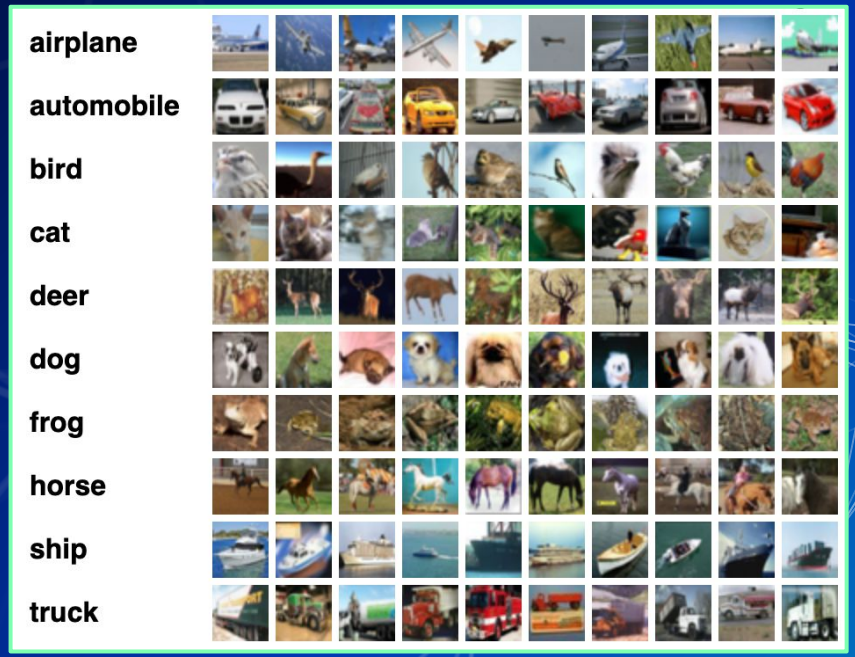
CIFAR-10

Arinola

Introduction: dataset, labels, etc.



10 classes

- The dataset contains 10 classes & contains six thousand images total
- Our training set contains five thousand images, and our test set contains one thousand images
- Our labels are one-hot encoded values 0-9
- For info check out this link: https://www.cs.toronto.edu/~kriz/cifar.html

Transfer Learning Model

https://www.tensorflow.org/tutorials/images/transfer_learning

Network Architecture

```
# 1. Model with ResNET Transfer Learning
resnet = tf.keras.applications.ResNet50(include_top=False, weights = 'imagenet', input_shape=(32, 32,3), classes = 10)

#Freezing the model:
resnet.trainable = False
```

```
flattened = tf.keras.layers.Flatten()(resnet.output)
dropped = tf.keras.layers.Dropout(0.2)(flattened)
fc1 = tf.keras.layers.Dense(1024, activation='relu', name = 'AddedDense1')(dropped)
fc2 = tf.keras.layers.Dense(10, activation = 'softmax', name = 'AddedDense2')(fc1)
model = tf.keras.models.Model(inputs = resnet.input, outputs = fc2)

model.summary()
```

First line: importing ResNET (Residual Neural Network)
Resnet.trainable = False: freezing the ResNET model's layers
fc1/fc2: adding two fully connected layers (with trainable parameters)

Network Architecture

Other Model

```
# 1. Make a Model
model = models.Sequential()

model.add(layers.Conv2D(32, (2, 2), input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))

model.add(layers.Conv2D(128, (2, 2)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))

model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

Hyper-Parameters

hyper-parameter	value
	learning model)

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

history5 = model.fit(datagen4.flow(train_images, Y_train, batch_size=32), epochs=5, batch_size=128)
history6 = model.fit(datagen5.flow(train_images, Y_train, batch_size=32), epochs=5, batch_size=128)
history2 = model.fit(datagen.flow(train_images, Y_train, batch_size=32), epochs=5, batch_size=128)
history3 = model.fit(datagen2.flow(train_images, Y_train, batch_size=32), epochs=5, batch_size=128)
history1 = model.fit(train_images, Y_train, epochs=5, batch_size=128)

yhat_test = model.predict(test_images)
```

Accuracy Log

Model architecture	Final accuracy on training set
(1) 1 Conv2D(32), 1 Dense(10)/Softmax output	0.6264, 5 epochs
(2) *Adjusted the following on (1):* Added 1 Conv2D Added BatchNormalization() after both Conv2D Moved activation to after	0.7830, 5 epochs
(3) *Adjusted the following on (2):* Added 2 MaxPooling2D() layers	0.6774, 5 epochs
(4) *Adjusted the following on (3)* Added a Dropout(.2) before the first MaxPooling	
(5) *Adjusted the following on (4)* From above: moved the dropout to after the second Conv2D	0.6801, 5 epochs
(6) *Adjusted the following on (4)* Changed dropout rate to 0.7	0.5992, 5 epochs

Accuracy Log

Model architecture	Final accuracy on training set
(7) *Scratched out everything from before and made a new code* `3 Conv2D (32, (2,2)), 1 MaxPooling (3,3), 3 Conv2D (64, (3,3)), 1 MaxPooling (3,3), Flatten, 2 Dense`	0.8201, 10 epochs
(8) *Adjusted the following on (3):* Added another Conv2D Got rid of the two MaxPooling2D	0.9359, 10 epochs
(9) *Adjusted the following on (1):* Changed first parameter on both Conv2d to 64	0.7561, 10 epochs
(10) *Adjusted the following on (1):* Changed first parameter on both Conv2d to 128	0.7960, 10 epochs
(11) *Adjusted the following on (2)* Increased epochs to 15	0.9572, 15 epochs
(12) *Adjusted the following on (10)* Also fit to horizontally flipped images	0.9196, 15 epochs

Accuracy Log

Model architecture	Final accuracy on training set
(13) 3 Conv2D layers, 2 BatchNorms & Activations, one Dense layer Trained on the images, horizontal flip, and random rotation	0.7652, but 0.6775 on test set
(14) The same as (13) but also trained on vertical flip	0.9875, but 0.629 on test set (0.9748/0.6583 with 10 epochs each)

Model Performance on Training Set (loss) & Test Set (training loss, test accuracy)

Model	Accuracy on Training Set	Accuracy on Test Set
Transfer Learning Model (ResNET)	0.7020	0.6263
		0.7238

```
import numpy as np
# 5. Plot y=x, comparing the true values to the predicted values
yhat_test_plot = tf.argmax(yhat_test, axis=1) # source: https://stackoverflo
y_test_plot = tf.argmax(Y_test, axis=1)
#plt.scatter(yhat_test_plot, y_test_plot)
print(f"accuracy of test set: {np.mean(yhat_test_plot - y_t

accuracy of test set: 0.6263
```

Accuracy of Test Set on the ResNET model

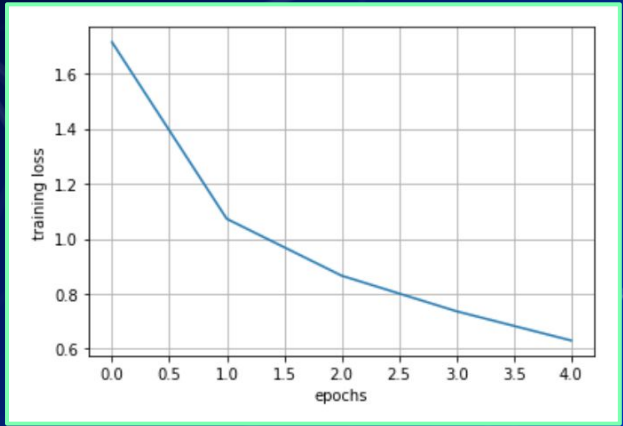
Accuracy of Test Set on the other model

```
import numpy as np
yhat_test_plot = tf.argmax(yhat_test, axis=1) # source: https://stackoverflo
y_test_plot = tf.argmax(Y_test, axis=1)
#plt.scatter(yhat_test_plot, y_test_plot)
print(f"accuracy of test set: {np.mean(yhat_test_plot - y_test_plot == 0)}")

accuracy of test set: 0.7238
```

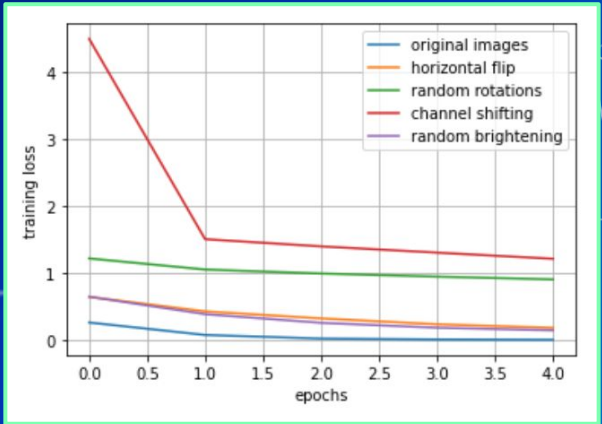
Error Throughout Training

one of our first models



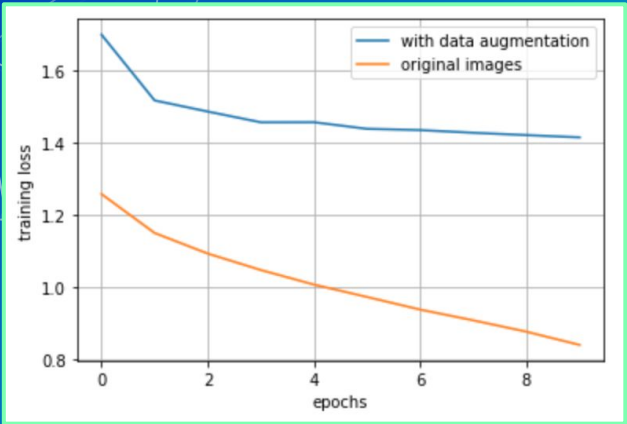
After 4 epochs: >0.6 training loss

most updated model



After 4 epochs: 0.0049 training loss

Transfer Learning model



After 8 epochs: 0.8385 training loss

Challenges & Solutions

Problem	Solution
Trouble plotting model • Error: Shapes (None, 1) and (None, 31, 31, 10) are incompatible	We remembered to add model.add(layers.Flatten()) into our model.
Epochs/tests were taking too long to load	Made copies + chart (multitasking) • Avoid losing track • Updates
Trouble with getting transfer learning to work	Worked with teacher • Rewrote code (ex. Condensed 5 history into 1, simplified data augmentation)

Conclusion

1. Introduction
   a. The data set: 10 classes and 6,000 images. The training set: 5,000 images. The test set: 1,000 images.
   b. We created a Model with ResNet Transfer learning
2. To-Do list
   a. We wanted to add convolutional layers, batch normalization, and dropout. We also wanted data argumentation.
3. Method of working
   a. Initially we created one Google Colab document, and had one person edit (while they shared their screen on Zoom) we gave suggestions as they coded. Later, we broke off and coded separately then got back together and shared code via screen share. .
4. Hyper-Parameters
   a. Using the adam optimizer we performed a stochastic gradient descent. We used the average learning rate of .001. Batch size: 128.

Conclusion

1. Model performance on training set:
   a. One model used standard layers, the other model used transfer learning with the ResNet model. When we ran both and decided that the transfer learning model was more accurate (70.2% accuracy). In the end we used model 2.
2. Error throughout training
   a. On the first model, after 4 epochs the training loss was 0.6
   b. The next model, after 4 epochs the training loss was .0049
   c. The transfer learning model, after 4 epochs the training loss was 0.8385
3. We had challenges with:
   a. Plotting the model (errors occurred) solution: model.add(layers.Flatten())
   b. Epochs/test (time consuming) solution: multitasking
   c. Transfer learning (we struggled to get it to work) solution: rewrote code
4. Links:
   a. https://www.cs.toronto.edu/~kriz/cifar.html