Keiland Pullen

CSC 578 NN&DL Spring, 2022

HW7: Image Classification using a CNN

This code is slightly modified from the TensorFlow tutorial Convolutional Neural Network (CNN) for the purpose of our homework. The code first downloads the data, the CIFAR-10 dataset and partitions the training set into training and validation sets. Then the code builds a CNN network and trains the network with the training set. Finally the code evaluates the network performance using the validation set.

Note that there are **three places** in the code, indicated with **IMPORTANT**, where you have to choose the syntax that works for the version of TensorFlow (1 or 2) installed on your platform.

Import Tensorflow

IMPORTANT (1) Uncomment either import line(s) for the version of TensorFlow (TF1 or TF2) of your platform.

In [214…

```
import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf
print(tf.__version__)  # check the TF version!
```

```
2.8.0
```

In [215…

```
# For TF version 2 (just one line)
from tensorflow.keras import datasets, layers, models

# For TF version 1 (need both lines)
# from tensorflow import keras
# from keras import datasets, layers, models
```

Download and prepare the CIFAR10 dataset¶

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is (pre-)divided into 50,000 training images and 10,000 testing images.

In [216…

```
# Download the data from the repository site.
(train_all_images, train_all_labels), (test_images, test_labels) = datasets.cifar10.loa
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

```
~\AppData\Local\Temp/ipykernel_17924/907951196.py in <module>
      1 # Download the data from the repository site.
----> 2 (train_all_images, train_all_labels), (test_images, test_labels) = datasets.cif
ar10.load_data()

~\anaconda3\lib\site-packages\keras\datasets\cifar10.py in load_data()
     77   dirname = 'cifar-10-batches-py'
     78   origin = 'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz'
---> 79   path = get_file(
     80       dirname,
     81       origin=origin,

~\anaconda3\lib\site-packages\keras\utils\data_utils.py in get_file(fname, origin, unta
r, md5_hash, file_hash, cache_subdir, hash_algorithm, extract, archive_format, cache_di
r)
    246     # File found; verify integrity if a hash was provided.
    247     if file_hash is not None:
--> 248       if not validate_file(fpath, file_hash, algorithm=hash_algorithm):
    249         io_utils.print_msg(
    250             'A local file was found, but it seems to be '

~\anaconda3\lib\site-packages\keras\utils\data_utils.py in validate_file(fpath, file_has
h, algorithm, chunk_size)
    360   hasher = _resolve_hasher(algorithm, file_hash)
    361 
--> 362   if str(_hash_file(fpath, hasher, chunk_size)) == str(file_hash):
    363     return True
    364   else:

~\anaconda3\lib\site-packages\keras\utils\data_utils.py in _hash_file(fpath, algorithm,
 chunk_size)
    339   with open(fpath, 'rb') as fpath_file:
    340     for chunk in iter(lambda: fpath_file.read(chunk_size), b''):
--> 341       hasher.update(chunk)
    342 
    343   return hasher.hexdigest()

KeyboardInterrupt:
```

In [ ]:

```
# !! DO NOT REMOVE THIS LINE !!
# Delete test_labels (by making it an empty list) so that we don't accidentally
# use it in the code.
#test_labels = []

# Then split the training set ('train_all') into two subsets: train and
# validation.  After that, we have 3 subsets: train, validation and test.
from sklearn.model_selection import train_test_split

# 80% train, 20% validation, and by using stratefied sampling.
train_images, valid_images, train_labels, valid_labels \
  = train_test_split(train_all_images, train_all_labels,
                     stratify=train_all_labels, test_size=0.2)
```

In [ ]:

```
# Normalize pixel values of images to be between 0 and 1
train_images, valid_images, test_images \
  = train_images / 255.0, valid_images / 255.0, test_images / 255.0
```

In [ ]:

```
test_labels
```

In [ ]:

```
train_labels
```

In [ ]:

```
valid_labels
```

Verify the data

To verify that the dataset looks correct, plot the first 10 images from the training set and display the class name below each image.

In [ ]:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

Create a convolutional network

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size, where color_channels refers to (R,G,B). The format of CIFAR images is 32 * 32 pixels, so the input shape is (32, 32, 3). The output layer has 10 nodes, corresponding to the number of categories of the images.

In this code, the activation function of the output layer is specified to be softmax for the purpose of aligning the two versions of TensorFlow (TF1 and TF2; in particular to make TF2 compatible with TF1's 'sparse_categorical_crossentropy' loss function).

In [ ]:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # As noted above
```

Verify the model

In [ ]:

```
model.summary()
```

Compile the model

IMPORTANT (2) Uncomment either loss function for the version of TensorFlow (TF1 or TF2) of your platform.

In [ ]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

Train the model

In [ ]:
```
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

Evaluate the model

IMPORTANT (3) Uncomment either syntax for the version of TensorFlow (TF1 or TF2) of your platform.

In [ ]:
```
plt.plot(history.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

TO DO (by you): Make Predictions

Apply the learned network to **'test_images'** and generate predictions.

Look at the code from HW#4 or other tutorial code for the syntax. You should generate predictions and create/write a KAGGLE submission file.

First Model

This model was built as a baseline to become familiar with the tools and project. Extra layers were added to this model to see the effect on its accuracy.

In [ ]:
```
model_1 = models.Sequential()
model_1.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model_1.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model_1.add(layers.MaxPooling2D((2, 2)))
model_1.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_1.add(layers.Conv2D(64, (3, 3), activation='relu'))

model_1.add(layers.MaxPooling2D((2, 2)))
model_1.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_1.add(layers.Conv2D(64, (3, 3), activation='relu'))

model_1.add(layers.Flatten())
model_1.add(layers.Dense(64, activation='relu'))
model_1.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [ ]:

```
model_1.summary()
```

In [ ]:

```
model_1.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [ ]:

```
history_1 = model_1.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

In [ ]:

```
history_1.history.keys()
```

In [ ]:

```
plt.plot(history_1.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_1.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_1.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

In [ ]:

```
print(valid_acc)
```

In [ ]:

```
predictions = model_1.predict(valid_images)
```

In [ ]:

```
predictions[0]
```

In [ ]:

```
np.argmax(predictions[0])
```

In [ ]:

```
# len(test_labels)

# len(valid_labels)
```

In [ ]:
```
valid_labels

#indices = np.random.choice(range(len(valid_labels)), replace=False)
#valid_labels=np.array(valid_labels)[indices.astype(int)]

valid_labels
```

In [ ]:
```
len(test_images)
```

In [ ]:
```
#test_labels[0]

valid_labels[0]
```

In [ ]:
```
valid_labels = np.concatenate(valid_labels)
```

In [ ]:
```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

In [ ]:
```
i = 9
plt.figure(figsize=(6,3))
```

```python
plt.subplot(1,2,1)
#plot_image(i, predictions[i], test_labels, test_images)
plot_image(i, predictions[i], valid_labels, valid_images)
plt.subplot(1,2,2)
#plot_value_array(i, predictions[i],  test_labels)
plot_value_array(i, predictions[i],  valid_labels)
plt.show()
```

In [ ]:

```python
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], valid_labels, valid_images)
  # plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  # plot_value_array(i, predictions[i], test_labels)
  plot_value_array(i, predictions[i], valid_labels)
plt.tight_layout()
plt.show()
```

In [ ]:

Model 2 </h2>

A duplicate of the first model. Trying to understand why there was an increase in accuracy with no changes.

In [ ]:

```python
model_2 = models.Sequential()
model_2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))

model_2.add(layers.MaxPooling2D((2, 2)))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))

model_2.add(layers.MaxPooling2D((2, 2)))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))

model_2.add(layers.Flatten())
model_2.add(layers.Dense(64, activation='relu'))
model_2.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [ ]:

```python
model_2.summary()
```

In [ ]:

```python
model_2.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
```

```
            #loss='sparse_categorical_crossentropy', # For TF1
            metrics=['accuracy'])
```

In [ ]:

```
history_2 = model_2.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

In [ ]:

```
history_2.history.keys()
```

In [ ]:

```
plt.plot(history_2.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_2.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_2.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

In [ ]:

```
print(valid_acc)
```

In [ ]:

```
predictions_2 = model_2.predict(valid_images)
```

In [ ]:

```
valid_labels = np.concatenate(valid_labels)
```

In [ ]:

```
i = 17
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
#plot_image(i, predictions[i], test_labels, test_images)
plot_image(i, predictions_2[i], valid_labels, valid_images)
plt.subplot(1,2,2)
#plot_value_array(i, predictions[i],  test_labels)
plot_value_array(i, predictions_2[i],  valid_labels)
plt.show()
```

In [ ]:

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 6
num_cols = 4
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions_2[i], valid_labels, valid_images)
  # plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  # plot_value_array(i, predictions[i], test_labels)
```

```
    plot_value_array(i, predictions_2[i], valid_labels)
plt.tight_layout()
plt.show()
```

In [ ]:

Model 3 - Removing layers

Several layers from the initial model were removed. Unfortunately, this resulted in overfitting.

In [79]:

```
model_3 = models.Sequential()
model_3.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(3
model_3.add(layers.MaxPooling2D((2, 2)))

model_3.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_3.add(layers.MaxPooling2D((2, 2)))

model_3.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_3.add(layers.Flatten())

model_3.add(layers.Dense(64, activation='relu'))
model_3.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [80]:

```
model_3.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_15 (Conv2D)          (None, 32, 32, 32)        896       
                                                                 
 max_pooling2d_8 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 conv2d_16 (Conv2D)          (None, 16, 16, 64)        18496     
                                                                 
 max_pooling2d_9 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 conv2d_17 (Conv2D)          (None, 8, 8, 64)          36928     
                                                                 
 flatten_4 (Flatten)         (None, 4096)              0         
                                                                 
 dense_8 (Dense)             (None, 64)                262208    
                                                                 
 dense_9 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 319,178
Trainable params: 319,178
Non-trainable params: 0
_________________________________________________________________
```

In [81]:

```
model_3.compile(optimizer='adam',
```

```
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [82]:

```
history_3 = model_3.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 55s 44ms/step - loss: 1.4706 - accuracy: 0.
4691 - val_loss: 1.1707 - val_accuracy: 0.5891
Epoch 2/10
1250/1250 [==============================] - 48s 38ms/step - loss: 1.0497 - accuracy: 0.
6279 - val_loss: 1.0067 - val_accuracy: 0.6491
Epoch 3/10
1250/1250 [==============================] - 54s 43ms/step - loss: 0.8824 - accuracy: 0.
6914 - val_loss: 0.9186 - val_accuracy: 0.6787
Epoch 4/10
1250/1250 [==============================] - 79s 63ms/step - loss: 0.7687 - accuracy: 0.
7330 - val_loss: 0.9376 - val_accuracy: 0.6788
Epoch 5/10
1250/1250 [==============================] - 67s 53ms/step - loss: 0.6880 - accuracy: 0.
7604 - val_loss: 0.8591 - val_accuracy: 0.7057
Epoch 6/10
1250/1250 [==============================] - 53s 42ms/step - loss: 0.6117 - accuracy: 0.
7858 - val_loss: 0.8327 - val_accuracy: 0.7172
Epoch 7/10
1250/1250 [==============================] - 54s 43ms/step - loss: 0.5388 - accuracy: 0.
8109 - val_loss: 0.8657 - val_accuracy: 0.7196
Epoch 8/10
1250/1250 [==============================] - 61s 48ms/step - loss: 0.4787 - accuracy: 0.
8307 - val_loss: 0.8865 - val_accuracy: 0.7164
Epoch 9/10
1250/1250 [==============================] - 56s 45ms/step - loss: 0.4168 - accuracy: 0.
8522 - val_loss: 0.9970 - val_accuracy: 0.7157
Epoch 10/10
1250/1250 [==============================] - 60s 48ms/step - loss: 0.3644 - accuracy: 0.
8709 - val_loss: 1.0529 - val_accuracy: 0.7085
```

In [83]:

```
history_3.history.keys()
```

Out[83]:

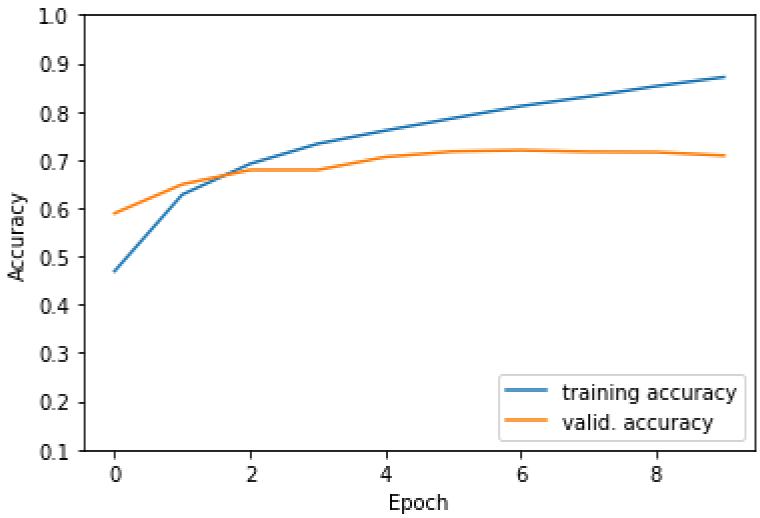
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [84]:

```
plt.plot(history_3.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_3.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_3.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 3s - loss: 1.0529 - accuracy: 0.7085 - 3s/epoch - 11ms/step
```

valid_accuracy=0.7085000276565552, valid_loss=1.0528604984283447



In [ ]:

Model 4 - Trial and error with layers and padding

Unfortunately, the final result for this model was overfitting.

In [86]:

```
model_4 = models.Sequential()
model_4.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(3
model_4.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_4.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_4.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_4.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_4.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_4.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_4.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_4.add(layers.Flatten())
model_4.add(layers.Dense(64, activation='relu'))
model_4.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [87]:

```
model_4.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_19 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_10 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_20 (Conv2D)          (None, 16, 16, 64)        18496
```

```
 conv2d_21 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_11 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 conv2d_22 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_23 (Conv2D)          (None, 8, 8, 64)          36928

 flatten_5 (Flatten)         (None, 4096)              0

 dense_10 (Dense)            (None, 64)                262208

 dense_11 (Dense)            (None, 10)                650

=================================================================
Total params: 402,282
Trainable params: 402,282
Non-trainable params: 0
_________________________________________________________________
```

In [88]:

```python
model_4.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [89]:

```python
history_4 = model_4.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 140s 112ms/step - loss: 1.5327 - accuracy:
0.4390 - val_loss: 1.2418 - val_accuracy: 0.5564
Epoch 2/10
1250/1250 [==============================] - 145s 116ms/step - loss: 1.0586 - accuracy:
0.6253 - val_loss: 0.9948 - val_accuracy: 0.6435
Epoch 3/10
1250/1250 [==============================] - 154s 123ms/step - loss: 0.8649 - accuracy:
0.6945 - val_loss: 0.8799 - val_accuracy: 0.6968
Epoch 4/10
1250/1250 [==============================] - 168s 135ms/step - loss: 0.7313 - accuracy:
0.7426 - val_loss: 0.8082 - val_accuracy: 0.7153
Epoch 5/10
1250/1250 [==============================] - 154s 123ms/step - loss: 0.6277 - accuracy:
0.7797 - val_loss: 0.7710 - val_accuracy: 0.7291
Epoch 6/10
1250/1250 [==============================] - 164s 131ms/step - loss: 0.5333 - accuracy:
0.8120 - val_loss: 0.7993 - val_accuracy: 0.7350
Epoch 7/10
1250/1250 [==============================] - 172s 137ms/step - loss: 0.4499 - accuracy:
0.8411 - val_loss: 0.8470 - val_accuracy: 0.7326
Epoch 8/10
1250/1250 [==============================] - 166s 132ms/step - loss: 0.3878 - accuracy:
0.8597 - val_loss: 0.8631 - val_accuracy: 0.7295
Epoch 9/10
1250/1250 [==============================] - 156s 125ms/step - loss: 0.3177 - accuracy:
0.8859 - val_loss: 0.9778 - val_accuracy: 0.7322
Epoch 10/10
```

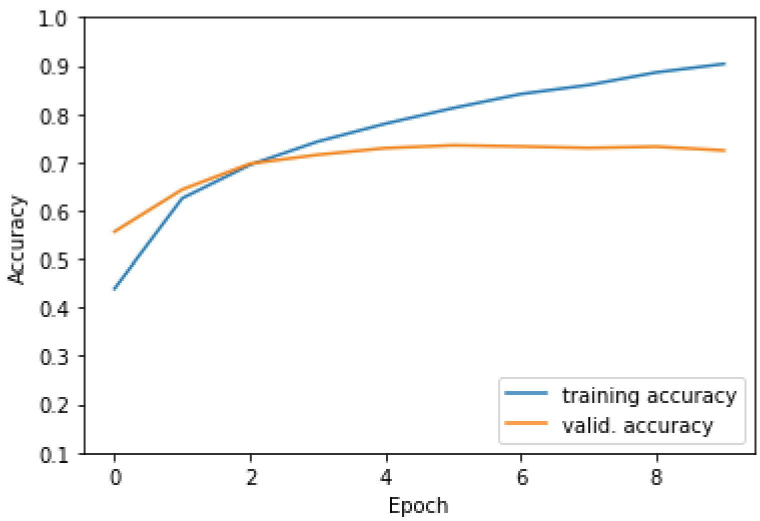
```
1250/1250 [==============================] - 155s 124ms/step - loss: 0.2719 - accuracy:
0.9034 - val_loss: 1.0310 - val_accuracy: 0.7243
```

In [91]:
```
history_4.history.keys()
```

Out[91]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [92]:
```
plt.plot(history_4.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_4.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_4.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 8s - loss: 1.0310 - accuracy: 0.7243 - 8s/epoch - 25ms/step
valid_accuracy=0.7243000268936157, valid_loss=1.0310277938842773
```



In [95]:
```
print(valid_acc)
```

```
0.7243000268936157
```

In [ ]:
```

```

Model 5 - Adjusting layers and experimenting with Spatial Dropout and Strides

The final result appearing to be a well fitting model, according to the plot. Unfortunately the accuracy of this model was below the accuracy of the initial model.

In [107…
```
model_5 = models.Sequential()
model_5.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(3
```

```python
model_5.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_5.add(layers.SpatialDropout2D(0.25))
model_5.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_5.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_5.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_5.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_5.add(layers.SpatialDropout2D(0.25))
model_5.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_5.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_5.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_5.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))

model_5.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_5.add(layers.Flatten())
model_5.add(layers.Dense(64, activation='relu'))
model_5.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [108…

```python
model_5.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_43 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_44 (Conv2D)          (None, 32, 32, 32)        9248

 spatial_dropout2d_8 (Spatia  (None, 32, 32, 32)       0
 lDropout2D)

 max_pooling2d_13 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_45 (Conv2D)          (None, 16, 16, 64)        18496

 conv2d_46 (Conv2D)          (None, 16, 16, 64)        36928

 conv2d_47 (Conv2D)          (None, 16, 16, 64)        36928

 spatial_dropout2d_9 (Spatia  (None, 16, 16, 64)       0
 lDropout2D)

 max_pooling2d_14 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 conv2d_48 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_49 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_50 (Conv2D)          (None, 8, 8, 64)          36928

 max_pooling2d_15 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)
```

```
 flatten_6 (Flatten)         (None, 1024)              0

 dense_12 (Dense)            (None, 64)                65600

 dense_13 (Dense)            (None, 10)                650

=================================================================
Total params: 279,530
Trainable params: 279,530
Non-trainable params: 0
_________________________________________________________________
```

In [109…

```
model_5.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [110…

```
history_5 = model_5.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 205s 163ms/step - loss: 1.7676 - accuracy:
0.3436 - val_loss: 1.4596 - val_accuracy: 0.4627
Epoch 2/10
1250/1250 [==============================] - 197s 158ms/step - loss: 1.3944 - accuracy:
0.4936 - val_loss: 1.2639 - val_accuracy: 0.5388
Epoch 3/10
1250/1250 [==============================] - 199s 159ms/step - loss: 1.2300 - accuracy:
0.5576 - val_loss: 1.0964 - val_accuracy: 0.6068
Epoch 4/10
1250/1250 [==============================] - 191s 152ms/step - loss: 1.1189 - accuracy:
0.6012 - val_loss: 1.1101 - val_accuracy: 0.6061
Epoch 5/10
1250/1250 [==============================] - 203s 163ms/step - loss: 1.0374 - accuracy:
0.6312 - val_loss: 0.9832 - val_accuracy: 0.6519
Epoch 6/10
1250/1250 [==============================] - 191s 153ms/step - loss: 0.9772 - accuracy:
0.6524 - val_loss: 0.9679 - val_accuracy: 0.6530
Epoch 7/10
1250/1250 [==============================] - 195s 156ms/step - loss: 0.9212 - accuracy:
0.6754 - val_loss: 0.9713 - val_accuracy: 0.6585
Epoch 8/10
1250/1250 [==============================] - 175s 140ms/step - loss: 0.8827 - accuracy:
0.6883 - val_loss: 0.9228 - val_accuracy: 0.6705
Epoch 9/10
1250/1250 [==============================] - 167s 134ms/step - loss: 0.8480 - accuracy:
0.6995 - val_loss: 0.8886 - val_accuracy: 0.6902
Epoch 10/10
1250/1250 [==============================] - 167s 134ms/step - loss: 0.8157 - accuracy:
0.7079 - val_loss: 0.8878 - val_accuracy: 0.6845
```

In [111…

```
history_5.history.keys()
```

Out[111…

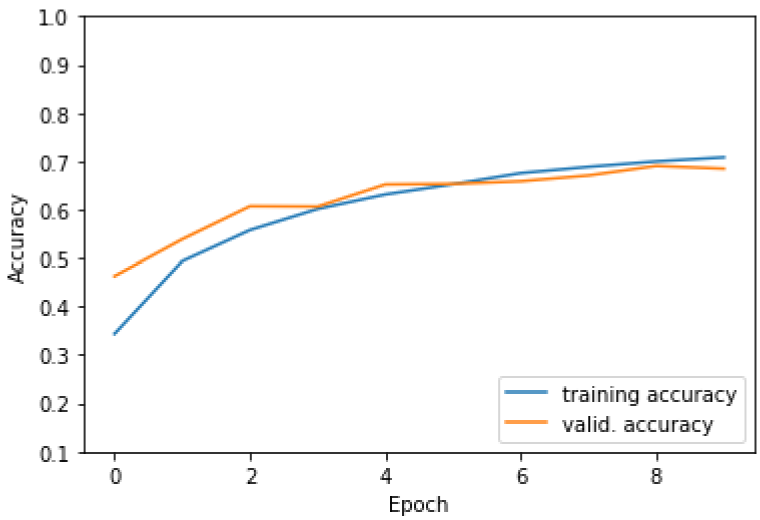
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [112…

```
plt.plot(history_5.history['accuracy'], label='training accuracy') # For TF2
```

```
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_5.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_5.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 8s - loss: 0.8878 - accuracy: 0.6845 - 8s/epoch - 26ms/step
valid_accuracy=0.684499979019165, valid_loss=0.8878217935562134
```



In [114…

```
test_loss, test_acc = model_5.evaluate(valid_images,  valid_labels)

print('\nTest accuracy:', test_acc)

print ('\n')
train_loss, train_acc = model_5.evaluate(train_images, train_labels, verbose = 2)
print('Training accuracy:', train_acc)
```

```
313/313 [==============================] - 9s 28ms/step - loss: 0.8878 - accuracy: 0.684
5

Test accuracy: 0.684499979019165


1250/1250 - 35s - loss: 0.6852 - accuracy: 0.7576 - 35s/epoch - 28ms/step
Training accuracy: 0.7575749754905701
```

In [ ]:

Model 6 - Experimenting with layers, Spatial Dropout and Strides

More layers were added to this model in addition to additional Epochs. Unfoturnately, the final result was a poor model. This particular model fails.

In [115…

```python
model_6 = models.Sequential()
model_6.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(3
model_6.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_6.add(layers.SpatialDropout2D(0.2))
model_6.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_6.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_6.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_6.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_6.add(layers.SpatialDropout2D(0.2))
model_6.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_6.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_6.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_6.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_6.add(layers.SpatialDropout2D(0.2))
model_6.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_6.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_6.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_6.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_6.add(layers.SpatialDropout2D(0.2))
model_6.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_6.add(layers.Flatten())
model_6.add(layers.Dense(64, activation='relu'))
model_6.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [116…

```python
model_6.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_51 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_52 (Conv2D)          (None, 32, 32, 32)        9248

 spatial_dropout2d_10 (Spati  (None, 32, 32, 32)       0
 alDropout2D)

 max_pooling2d_16 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_53 (Conv2D)          (None, 16, 16, 64)        18496

 conv2d_54 (Conv2D)          (None, 16, 16, 64)        36928

 conv2d_55 (Conv2D)          (None, 16, 16, 64)        36928

 spatial_dropout2d_11 (Spati  (None, 16, 16, 64)       0
 alDropout2D)

 max_pooling2d_17 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)
```

```
 conv2d_56 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_57 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_58 (Conv2D)          (None, 8, 8, 64)          36928

 spatial_dropout2d_12 (Spati  (None, 8, 8, 64)         0
 alDropout2D)

 max_pooling2d_18 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 conv2d_59 (Conv2D)          (None, 4, 4, 128)         73856

 conv2d_60 (Conv2D)          (None, 4, 4, 128)         147584

 conv2d_61 (Conv2D)          (None, 4, 4, 128)         147584

 spatial_dropout2d_13 (Spati  (None, 4, 4, 128)        0
 alDropout2D)

 max_pooling2d_19 (MaxPoolin  (None, 2, 2, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 512)               0

 dense_14 (Dense)            (None, 64)                32832

 dense_15 (Dense)            (None, 10)                650

=================================================================
Total params: 615,786
Trainable params: 615,786
Non-trainable params: 0
_________________________________________________________________
```

In [117…

```
model_6.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [118…

```
history_6 = model_6.fit(train_images, train_labels, epochs=15,
                    validation_data=(valid_images, valid_labels))
```

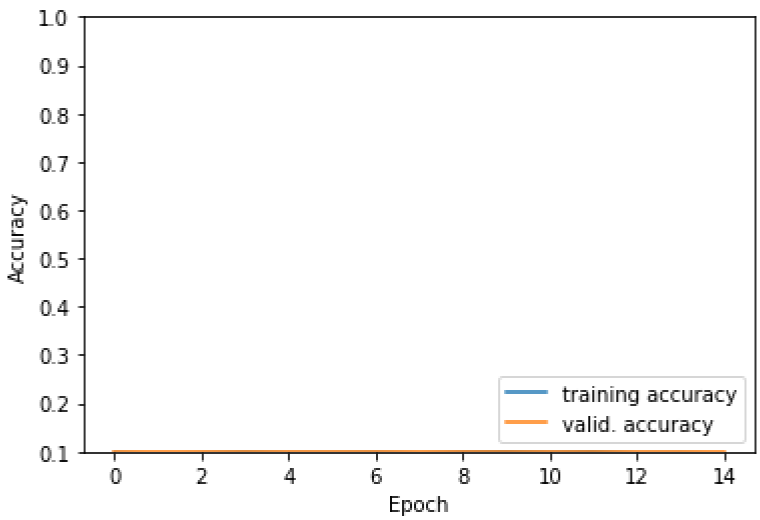
```
Epoch 1/15
1250/1250 [==============================] - 176s 140ms/step - loss: 2.3030 - accuracy:
0.0975 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 2/15
1250/1250 [==============================] - 180s 144ms/step - loss: 2.3028 - accuracy:
0.0954 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 3/15
1250/1250 [==============================] - 180s 144ms/step - loss: 2.3028 - accuracy:
0.0967 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 4/15
1250/1250 [==============================] - 180s 144ms/step - loss: 2.3028 - accuracy:
0.0984 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 5/15
1250/1250 [==============================] - 181s 145ms/step - loss: 2.3027 - accuracy:
```

```
0.0970 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 6/15
1250/1250 [==============================] - 180s 144ms/step - loss: 2.3028 - accuracy:
0.0971 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 7/15
1250/1250 [==============================] - 182s 145ms/step - loss: 2.3028 - accuracy:
0.0978 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 8/15
1250/1250 [==============================] - 180s 144ms/step - loss: 2.3028 - accuracy:
0.0962 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 9/15
1250/1250 [==============================] - 185s 148ms/step - loss: 2.3028 - accuracy:
0.0983 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 10/15
1250/1250 [==============================] - 181s 145ms/step - loss: 2.3028 - accuracy:
0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 11/15
1250/1250 [==============================] - 187s 149ms/step - loss: 2.3028 - accuracy:
0.0979 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 12/15
1250/1250 [==============================] - 192s 154ms/step - loss: 2.3028 - accuracy:
0.0992 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 13/15
1250/1250 [==============================] - 184s 147ms/step - loss: 2.3028 - accuracy:
0.0965 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 14/15
1250/1250 [==============================] - 181s 145ms/step - loss: 2.3028 - accuracy:
0.0970 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 15/15
1250/1250 [==============================] - 181s 145ms/step - loss: 2.3028 - accuracy:
0.0956 - val_loss: 2.3026 - val_accuracy: 0.1000
```

In [119…

```
plt.plot(history_6.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_6.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_6.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 9s - loss: 2.3026 - accuracy: 0.1000 - 9s/epoch - 30ms/step
valid_accuracy=0.10000000149011612, valid_loss=2.302598714828491
```



In [120…

```
test_loss, test_acc = model_6.evaluate(valid_images,  valid_labels)

print('\nTest accuracy:', test_acc)

print ('\n')
train_loss, train_acc = model_6.evaluate(train_images, train_labels, verbose = 2)
print('Training accuracy:', train_acc)
```

```
313/313 [==============================] - 11s 34ms/step - loss: 2.3026 - accuracy: 0.10
00

Test accuracy: 0.10000000149011612


1250/1250 - 38s - loss: 2.3026 - accuracy: 0.1000 - 38s/epoch - 30ms/step
Training accuracy: 0.10000000149011612
```

In [ ]:

```

```

Model 7 - Experiment with Batch Normalization

In this model, Batch Normalization was added in addition to reducing the number of layers and adjusting the Spatial Dropout. The final result appears to be a decent model with an accuracy of 74%.

In [122…

```
from keras.layers import BatchNormalization
```

In [123…

```
model_7 = models.Sequential()
model_7.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(3
model_7.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_7.add(layers.SpatialDropout2D(0.25))
model_7.add(BatchNormalization())
model_7.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_7.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_7.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_7.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
```

```
model_7.add(layers.SpatialDropout2D(0.25))
model_7.add(BatchNormalization())
model_7.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_7.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_7.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_7.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_7.add(layers.SpatialDropout2D(0.25))
model_7.add(BatchNormalization())
model_7.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_7.add(layers.Flatten())
model_7.add(layers.Dense(64, activation='relu'))
model_7.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [124…

```
model_7.summary()
```

```
Model: "sequential_17"

_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_64 (Conv2D)          (None, 32, 32, 32)        896       
                                                                 
 conv2d_65 (Conv2D)          (None, 32, 32, 32)        9248      
                                                                 
 spatial_dropout2d_15 (Spati  (None, 32, 32, 32)       0         
 alDropout2D)                                                    
                                                                 
 batch_normalization (BatchN  (None, 32, 32, 32)       128       
 ormalization)                                                   
                                                                 
 max_pooling2d_20 (MaxPoolin  (None, 16, 16, 32)       0         
 g2D)                                                            
                                                                 
 conv2d_66 (Conv2D)          (None, 16, 16, 64)        18496     
                                                                 
 conv2d_67 (Conv2D)          (None, 16, 16, 64)        36928     
                                                                 
 conv2d_68 (Conv2D)          (None, 16, 16, 64)        36928     
                                                                 
 spatial_dropout2d_16 (Spati  (None, 16, 16, 64)       0         
 alDropout2D)                                                    
                                                                 
 batch_normalization_1 (Batc  (None, 16, 16, 64)       256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_21 (MaxPoolin  (None, 8, 8, 64)         0         
 g2D)                                                            
                                                                 
 conv2d_69 (Conv2D)          (None, 8, 8, 64)          36928     
                                                                 
 conv2d_70 (Conv2D)          (None, 8, 8, 64)          36928     
                                                                 
 conv2d_71 (Conv2D)          (None, 8, 8, 64)          36928     
                                                                 
 spatial_dropout2d_17 (Spati  (None, 8, 8, 64)         0         
 alDropout2D)                                                    
```

```
 batch_normalization_2 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 max_pooling2d_22 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 flatten_8 (Flatten)         (None, 1024)              0

 dense_16 (Dense)            (None, 64)                65600

 dense_17 (Dense)            (None, 10)                650

=================================================================
Total params: 280,170
Trainable params: 279,850
Non-trainable params: 320
_________________________________________________________________
```

In [125…

```
model_7.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

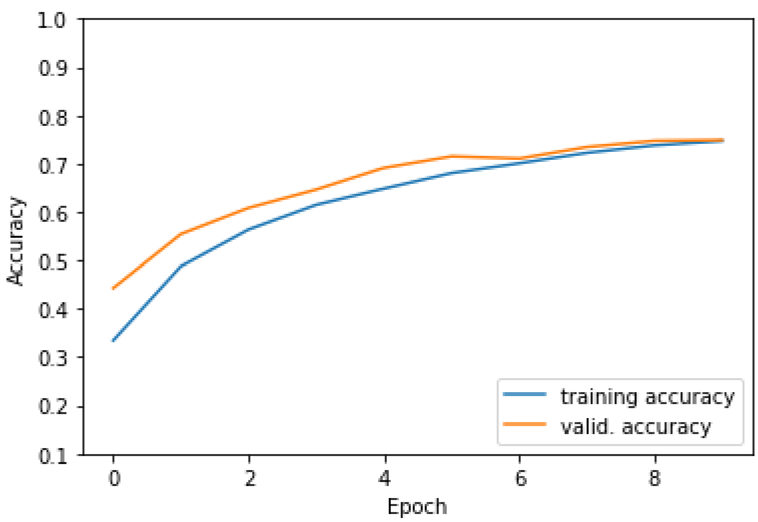
In [126…

```
history_7 = model_7.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 174s 139ms/step - loss: 1.8080 - accuracy:
0.3345 - val_loss: 1.5204 - val_accuracy: 0.4427
Epoch 2/10
1250/1250 [==============================] - 181s 145ms/step - loss: 1.4243 - accuracy:
0.4876 - val_loss: 1.2131 - val_accuracy: 0.5540
Epoch 3/10
1250/1250 [==============================] - 181s 144ms/step - loss: 1.2179 - accuracy:
0.5633 - val_loss: 1.1156 - val_accuracy: 0.6081
Epoch 4/10
1250/1250 [==============================] - 179s 144ms/step - loss: 1.0809 - accuracy:
0.6144 - val_loss: 0.9936 - val_accuracy: 0.6460
Epoch 5/10
1250/1250 [==============================] - 194s 155ms/step - loss: 0.9944 - accuracy:
0.6481 - val_loss: 0.8790 - val_accuracy: 0.6910
Epoch 6/10
1250/1250 [==============================] - 190s 152ms/step - loss: 0.9088 - accuracy:
0.6803 - val_loss: 0.8026 - val_accuracy: 0.7149
Epoch 7/10
1250/1250 [==============================] - 187s 150ms/step - loss: 0.8499 - accuracy:
0.7005 - val_loss: 0.8234 - val_accuracy: 0.7104
Epoch 8/10
1250/1250 [==============================] - 180s 144ms/step - loss: 0.7969 - accuracy:
0.7222 - val_loss: 0.7482 - val_accuracy: 0.7342
Epoch 9/10
1250/1250 [==============================] - 180s 144ms/step - loss: 0.7488 - accuracy:
0.7376 - val_loss: 0.7219 - val_accuracy: 0.7470
Epoch 10/10
1250/1250 [==============================] - 182s 145ms/step - loss: 0.7244 - accuracy:
0.7468 - val_loss: 0.7079 - val_accuracy: 0.7486
```

In [127…
```
plt.plot(history_7.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_7.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_7.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 9s - loss: 0.7079 - accuracy: 0.7486 - 9s/epoch - 29ms/step
valid_accuracy=0.7486000061035156, valid_loss=0.7078680396080017
```



In [128…
```
test_loss, test_acc = model_7.evaluate(valid_images,  valid_labels)

print('\nTest accuracy:', test_acc)

print ('\n')
train_loss, train_acc = model_7.evaluate(train_images, train_labels, verbose = 2)
print('Training accuracy:', train_acc)
```

```
313/313 [==============================] - 10s 31ms/step - loss: 0.7079 - accuracy: 0.74
86

Test accuracy: 0.7486000061035156


1250/1250 - 40s - loss: 0.5335 - accuracy: 0.8130 - 40s/epoch - 32ms/step
Training accuracy: 0.8130499720573425
```

In [ ]:

Model 8 - Experimenting with Optimizers

This model is built on top of model 7 and will experiment with several optimizers. The Adagrad optimizer fails as it causes the model performance to suffer. The Adadelta model also fails. The introduction of Adamax results in a very close fit for the data after the 8th Epoch, and its accuracy is stead at 74%.

In [129…

```
model_8 = models.Sequential()
model_8.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(3
model_8.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_8.add(layers.SpatialDropout2D(0.25))
model_8.add(BatchNormalization())
model_8.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_8.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_8.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_8.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_8.add(layers.SpatialDropout2D(0.25))
model_8.add(BatchNormalization())
model_8.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_8.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_8.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_8.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_8.add(layers.SpatialDropout2D(0.25))
model_8.add(BatchNormalization())
model_8.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_8.add(layers.Flatten())
model_8.add(layers.Dense(64, activation='relu'))
model_8.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [130…

```
model_8.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_72 (Conv2D)          (None, 32, 32, 32)        896       
                                                                 
 conv2d_73 (Conv2D)          (None, 32, 32, 32)        9248      
                                                                 
 spatial_dropout2d_18 (Spati  (None, 32, 32, 32)       0         
 alDropout2D)                                                    
                                                                 
 batch_normalization_3 (Batc  (None, 32, 32, 32)       128       
 hNormalization)                                                 
                                                                 
 max_pooling2d_23 (MaxPoolin  (None, 16, 16, 32)       0         
 g2D)                                                            
                                                                 
 conv2d_74 (Conv2D)          (None, 16, 16, 64)        18496     
                                                                 
 conv2d_75 (Conv2D)          (None, 16, 16, 64)        36928     
                                                                 
 conv2d_76 (Conv2D)          (None, 16, 16, 64)        36928     
                                                                 
 spatial_dropout2d_19 (Spati  (None, 16, 16, 64)       0         
 alDropout2D)                                                    
```

```
 batch_normalization_4 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_24 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 conv2d_77 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_78 (Conv2D)          (None, 8, 8, 64)          36928

 conv2d_79 (Conv2D)          (None, 8, 8, 64)          36928

 spatial_dropout2d_20 (Spati  (None, 8, 8, 64)         0
 alDropout2D)

 batch_normalization_5 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 max_pooling2d_25 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 flatten_9 (Flatten)         (None, 1024)              0

 dense_18 (Dense)            (None, 64)                65600

 dense_19 (Dense)            (None, 10)                650

=================================================================
Total params: 280,170
Trainable params: 279,850
Non-trainable params: 320
_________________________________________________________________
```

Adagrad Optimizer

In [131…

```
model_8.compile(optimizer='adagrad',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])

# Final Test Accuracy =  0.44539999961853O3
# Final Training accuracy = 0.45414999127388
```

In [132…

```
history_8 = model_8.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

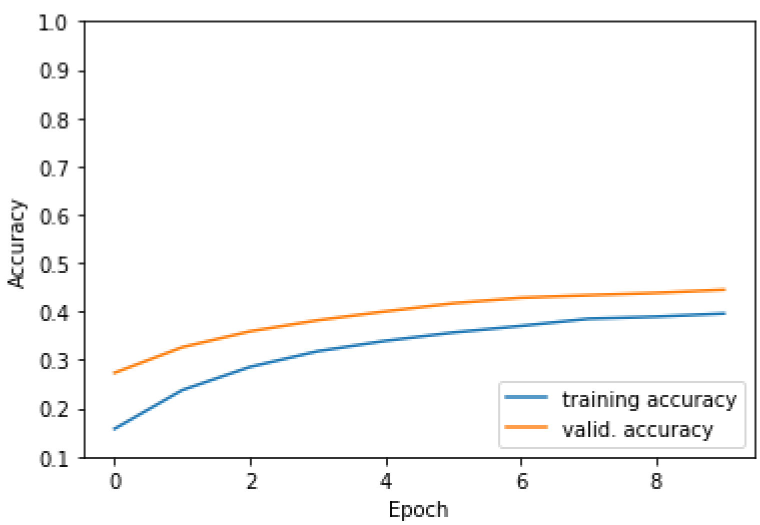
```
Epoch 1/10
1250/1250 [==============================] - 185s 148ms/step - loss: 2.3063 - accuracy:
0.1575 - val_loss: 2.0273 - val_accuracy: 0.2734
Epoch 2/10
1250/1250 [==============================] - 189s 151ms/step - loss: 2.0612 - accuracy:
0.2376 - val_loss: 1.8543 - val_accuracy: 0.3264
Epoch 3/10
1250/1250 [==============================] - 189s 151ms/step - loss: 1.9348 - accuracy:
0.2856 - val_loss: 1.7491 - val_accuracy: 0.3592
Epoch 4/10
1250/1250 [==============================] - 189s 151ms/step - loss: 1.8529 - accuracy:
```

```
0.3180 - val_loss: 1.6866 - val_accuracy: 0.3821
Epoch 5/10
1250/1250 [==============================] - 190s 152ms/step - loss: 1.7930 - accuracy:
0.3395 - val_loss: 1.6340 - val_accuracy: 0.4005
Epoch 6/10
1250/1250 [==============================] - 193s 154ms/step - loss: 1.7482 - accuracy:
0.3568 - val_loss: 1.5913 - val_accuracy: 0.4173
Epoch 7/10
1250/1250 [==============================] - 197s 158ms/step - loss: 1.7133 - accuracy:
0.3704 - val_loss: 1.5596 - val_accuracy: 0.4287
Epoch 8/10
1250/1250 [==============================] - 204s 163ms/step - loss: 1.6791 - accuracy:
0.3853 - val_loss: 1.5354 - val_accuracy: 0.4340
Epoch 9/10
1250/1250 [==============================] - 198s 158ms/step - loss: 1.6601 - accuracy:
0.3896 - val_loss: 1.5175 - val_accuracy: 0.4387
Epoch 10/10
1250/1250 [==============================] - 195s 156ms/step - loss: 1.6410 - accuracy:
0.3961 - val_loss: 1.4925 - val_accuracy: 0.4454
```

In [133…

```
plt.plot(history_8.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_8.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_8.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 9s - loss: 1.4925 - accuracy: 0.4454 - 9s/epoch - 28ms/step
valid_accuracy=0.4453999996185303, valid_loss=1.492461919784546
```



Adadelta Optimizer

In [136…

```
model_8.compile(optimizer='adadelta',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [137…

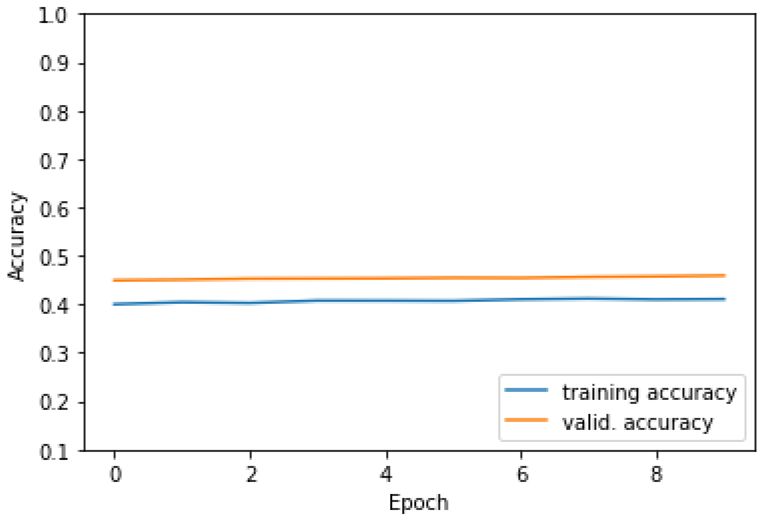
```
history_8 = model_8.fit(train_images, train_labels, epochs=10,
                        validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 174s 137ms/step - loss: 1.6229 - accuracy:
0.4005 - val_loss: 1.4882 - val_accuracy: 0.4499
Epoch 2/10
1250/1250 [==============================] - 176s 141ms/step - loss: 1.6228 - accuracy:
0.4047 - val_loss: 1.4866 - val_accuracy: 0.4512
Epoch 3/10
1250/1250 [==============================] - 180s 144ms/step - loss: 1.6227 - accuracy:
0.4030 - val_loss: 1.4815 - val_accuracy: 0.4533
Epoch 4/10
1250/1250 [==============================] - 180s 144ms/step - loss: 1.6198 - accuracy:
0.4080 - val_loss: 1.4805 - val_accuracy: 0.4537
Epoch 5/10
1250/1250 [==============================] - 204s 163ms/step - loss: 1.6129 - accuracy:
0.4078 - val_loss: 1.4788 - val_accuracy: 0.4541
Epoch 6/10
1250/1250 [==============================] - 190s 152ms/step - loss: 1.6109 - accuracy:
0.4074 - val_loss: 1.4767 - val_accuracy: 0.4552
Epoch 7/10
1250/1250 [==============================] - 181s 145ms/step - loss: 1.6094 - accuracy:
0.4106 - val_loss: 1.4736 - val_accuracy: 0.4549
Epoch 8/10
1250/1250 [==============================] - 188s 151ms/step - loss: 1.6079 - accuracy:
0.4121 - val_loss: 1.4686 - val_accuracy: 0.4569
Epoch 9/10
1250/1250 [==============================] - 190s 152ms/step - loss: 1.6115 - accuracy:
0.4103 - val_loss: 1.4663 - val_accuracy: 0.4581
Epoch 10/10
1250/1250 [==============================] - 191s 153ms/step - loss: 1.6000 - accuracy:
0.4110 - val_loss: 1.4630 - val_accuracy: 0.4597
```

In [138…

```
plt.plot(history_8.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_8.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_8.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 9s - loss: 1.4630 - accuracy: 0.4597 - 9s/epoch - 29ms/step
valid_accuracy=0.45969998836517334, valid_loss=1.4630039930343628
```



Adamax Optimizer

In [141…

```
model_8.compile(optimizer='adamax',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [142…

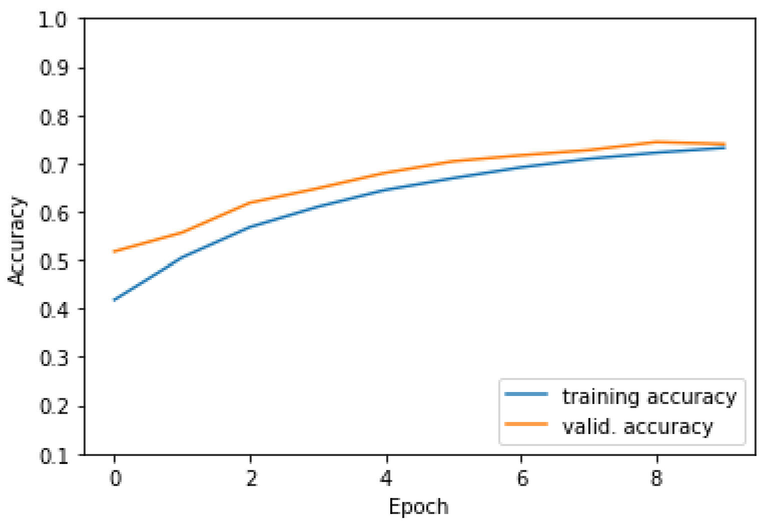
```
history_8 = model_8.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 192s 153ms/step - loss: 1.5985 - accuracy:
0.4182 - val_loss: 1.3284 - val_accuracy: 0.5175
Epoch 2/10
1250/1250 [==============================] - 181s 145ms/step - loss: 1.3654 - accuracy:
0.5052 - val_loss: 1.2312 - val_accuracy: 0.5565
Epoch 3/10
1250/1250 [==============================] - 181s 145ms/step - loss: 1.2031 - accuracy:
0.5676 - val_loss: 1.0539 - val_accuracy: 0.6180
Epoch 4/10
1250/1250 [==============================] - 181s 145ms/step - loss: 1.0917 - accuracy:
0.6096 - val_loss: 0.9752 - val_accuracy: 0.6478
Epoch 5/10
1250/1250 [==============================] - 180s 144ms/step - loss: 1.0022 - accuracy:
0.6446 - val_loss: 0.8990 - val_accuracy: 0.6801
Epoch 6/10
1250/1250 [==============================] - 181s 145ms/step - loss: 0.9326 - accuracy:
0.6691 - val_loss: 0.8351 - val_accuracy: 0.7040
Epoch 7/10
1250/1250 [==============================] - 181s 145ms/step - loss: 0.8764 - accuracy:
0.6914 - val_loss: 0.7992 - val_accuracy: 0.7163
Epoch 8/10
1250/1250 [==============================] - 181s 145ms/step - loss: 0.8280 - accuracy:
0.7090 - val_loss: 0.7812 - val_accuracy: 0.7270
Epoch 9/10
1250/1250 [==============================] - 187s 150ms/step - loss: 0.7884 - accuracy:
0.7218 - val_loss: 0.7428 - val_accuracy: 0.7438
Epoch 10/10
1250/1250 [==============================] - 191s 153ms/step - loss: 0.7538 - accuracy:
0.7318 - val_loss: 0.7362 - val_accuracy: 0.7396
```

In [143…

```
plt.plot(history_8.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_8.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_8.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 9s - loss: 0.7362 - accuracy: 0.7396 - 9s/epoch - 29ms/step
valid_accuracy=0.7396000027656555, valid_loss=0.7362483143806458
```



In [144…

```
test_loss, test_acc = model_2.evaluate(test_images,  test_labels)

print('\nTest accuracy:', valid_acc)

print ('\n')
train_loss, train_acc = model.evaluate(train_images, train_labels, verbose = 2)
print('Training accuracy:', train_acc)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0000e+00 - accuracy: 0.
0000e+00

Test accuracy: 0.7396000027656555


1250/1250 - 8s - loss: 0.5887 - accuracy: 0.7922 - 8s/epoch - 7ms/step
Training accuracy: 0.7921749949455261
```

In [146…

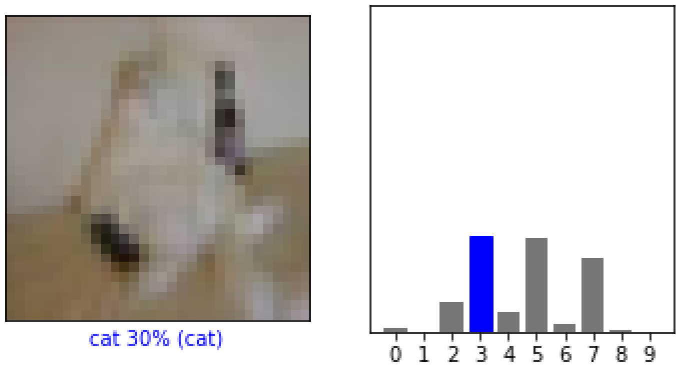
```
predictions_8 = model_8.predict(valid_images)
```

In [149…

```
#valid_labels = np.concatenate(valid_labels)

#valid_labels
```

In [152…

```
i = 21
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
#plot_image(i, predictions[i], test_labels, test_images)
plot_image(i, predictions_2[i], valid_labels, valid_images)
plt.subplot(1,2,2)
#plot_value_array(i, predictions[i],  test_labels)
plot_value_array(i, predictions_2[i],  valid_labels)
plt.show()
```



In [151…

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 6
num_cols = 4
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions_2[i], valid_labels, valid_images)
  # plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  # plot_value_array(i, predictions[i], test_labels)
  plot_value_array(i, predictions_2[i], valid_labels)
plt.tight_layout()
plt.show()
```

BEST FITTING MODEL

Model 10

For this model, the "Adam" optimizer was used, in addition to adjustments made by adding more layers, Batch Normalization, and Dropout. This model was tested on 10 and 15 Epochs.

The final result is a model with an accuracy of 80%. Unfortunately, there appears to be some overfitting in the early Epochs, but with additional Epochs, the area between the training and validation data may decrease.

In [183…

```
model_10 = models.Sequential()
model_10.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(
model_10.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Conv2D(32, (3, 3), padding ='same', activation='relu'))
model_10.add(layers.Dropout(0.1))
model_10.add(BatchNormalization())
model_10.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_10.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
```

```
model_10.add(layers.Dropout(0.1))
model_10.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_10.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Dropout(0.1))
model_10.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_10.add(layers.Flatten())
model_10.add(layers.Dense(128, activation='relu'))
model_10.add(BatchNormalization())
model_10.add(layers.Dropout(0.1))
model_10.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [184…

```
model_10.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [173…

```
history_10 = model_10.fit(train_images, train_labels, epochs=15,
                    validation_data=(valid_images, valid_labels))
```

```
Epoch 1/15
1250/1250 [==============================] - 329s 262ms/step - loss: 1.4625 - accuracy:
0.4789 - val_loss: 1.3672 - val_accuracy: 0.5144
Epoch 2/15
1250/1250 [==============================] - 332s 266ms/step - loss: 0.9488 - accuracy:
0.6648 - val_loss: 1.6568 - val_accuracy: 0.4793
Epoch 3/15
1250/1250 [==============================] - 333s 266ms/step - loss: 0.7762 - accuracy:
0.7312 - val_loss: 1.1284 - val_accuracy: 0.6595
Epoch 4/15
1250/1250 [==============================] - 335s 268ms/step - loss: 0.6772 - accuracy:
0.7694 - val_loss: 0.8895 - val_accuracy: 0.6960
Epoch 5/15
1250/1250 [==============================] - 332s 266ms/step - loss: 0.5947 - accuracy:
0.7984 - val_loss: 0.9126 - val_accuracy: 0.6957
Epoch 6/15
1250/1250 [==============================] - 337s 269ms/step - loss: 0.5215 - accuracy:
0.8224 - val_loss: 0.8137 - val_accuracy: 0.7240
Epoch 7/15
1250/1250 [==============================] - 337s 269ms/step - loss: 0.4642 - accuracy:
0.8422 - val_loss: 0.9060 - val_accuracy: 0.6970
Epoch 8/15
1250/1250 [==============================] - 333s 267ms/step - loss: 0.4086 - accuracy:
0.8594 - val_loss: 0.8081 - val_accuracy: 0.7381
Epoch 9/15
1250/1250 [==============================] - 333s 266ms/step - loss: 0.3607 - accuracy:
0.8750 - val_loss: 0.6932 - val_accuracy: 0.7837
Epoch 10/15
```

```
1250/1250 [==============================] - 335s 268ms/step - loss: 0.3152 - accuracy: 0.8915 - val_loss: 0.6889 - val_accuracy: 0.7835
Epoch 11/15
1250/1250 [==============================] - 333s 266ms/step - loss: 0.2783 - accuracy: 0.9040 - val_loss: 0.7120 - val_accuracy: 0.7770
Epoch 12/15
1250/1250 [==============================] - 333s 266ms/step - loss: 0.2509 - accuracy: 0.9121 - val_loss: 0.6087 - val_accuracy: 0.8050
Epoch 13/15
1250/1250 [==============================] - 333s 266ms/step - loss: 0.2250 - accuracy: 0.9227 - val_loss: 0.7350 - val_accuracy: 0.7693
Epoch 14/15
1250/1250 [==============================] - 372s 298ms/step - loss: 0.2042 - accuracy: 0.9292 - val_loss: 0.7470 - val_accuracy: 0.7781
Epoch 15/15
1250/1250 [==============================] - 401s 321ms/step - loss: 0.1868 - accuracy: 0.9361 - val_loss: 0.6612 - val_accuracy: 0.8042
```

In [185…

```
history_10 = model_10.fit(train_images, train_labels, epochs=10,
                          validation_data=(valid_images, valid_labels))
```

```
Epoch 1/10
1250/1250 [==============================] - 331s 264ms/step - loss: 1.3658 - accuracy: 0.5167 - val_loss: 1.2339 - val_accuracy: 0.5835
Epoch 2/10
1250/1250 [==============================] - 334s 267ms/step - loss: 0.8746 - accuracy: 0.6955 - val_loss: 0.8144 - val_accuracy: 0.7130
Epoch 3/10
1250/1250 [==============================] - 334s 267ms/step - loss: 0.7180 - accuracy: 0.7515 - val_loss: 0.8029 - val_accuracy: 0.7269
Epoch 4/10
1250/1250 [==============================] - 336s 269ms/step - loss: 0.6113 - accuracy: 0.7888 - val_loss: 0.6781 - val_accuracy: 0.7725
Epoch 5/10
1250/1250 [==============================] - 337s 270ms/step - loss: 0.5286 - accuracy: 0.8196 - val_loss: 0.5918 - val_accuracy: 0.8000
Epoch 6/10
1250/1250 [==============================] - 336s 269ms/step - loss: 0.4498 - accuracy: 0.8474 - val_loss: 0.6852 - val_accuracy: 0.7716
Epoch 7/10
1250/1250 [==============================] - 334s 267ms/step - loss: 0.3890 - accuracy: 0.8675 - val_loss: 0.6669 - val_accuracy: 0.7859
Epoch 8/10
1250/1250 [==============================] - 334s 267ms/step - loss: 0.3264 - accuracy: 0.8866 - val_loss: 0.5784 - val_accuracy: 0.8107
Epoch 9/10
1250/1250 [==============================] - 337s 269ms/step - loss: 0.2791 - accuracy: 0.9032 - val_loss: 0.7755 - val_accuracy: 0.7685
Epoch 10/10
1250/1250 [==============================] - 342s 274ms/step - loss: 0.2447 - accuracy: 0.9137 - val_loss: 0.6264 - val_accuracy: 0.8040
```

In [186…

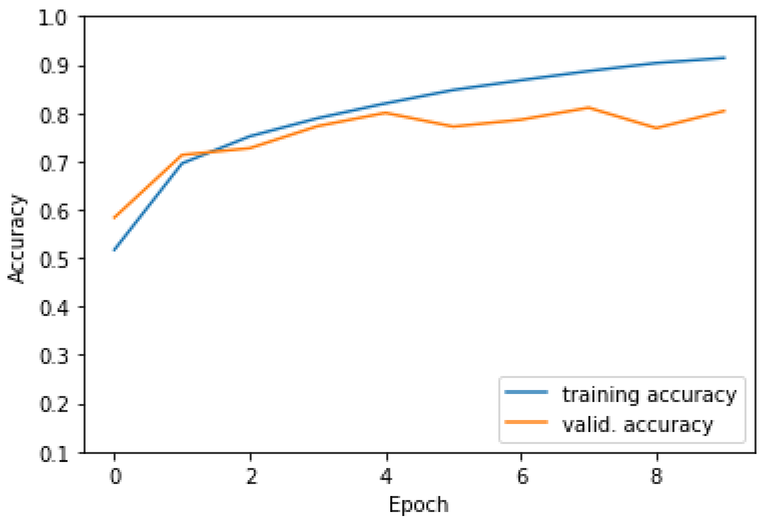
```
test_loss, test_acc = model_10.evaluate(test_images,  test_labels)

print('\nTest accuracy:', valid_acc)

print ('\n')
train_loss, train_acc = model_10.evaluate(train_images, train_labels, verbose = 2)
print('Training accuracy:', train_acc)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0000e+00 - accuracy: 0.
0000e+00

Test accuracy: 0.8041999936103821


1250/1250 - 67s - loss: 0.2064 - accuracy: 0.9313 - 67s/epoch - 53ms/step
Training accuracy: 0.9313499927520752
```

In [187…

```python
plt.plot(history_10.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_10.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_10.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 16s - loss: 0.6264 - accuracy: 0.8040 - 16s/epoch - 50ms/step
valid_accuracy=0.8040000200271606, valid_loss=0.626409649848938
```

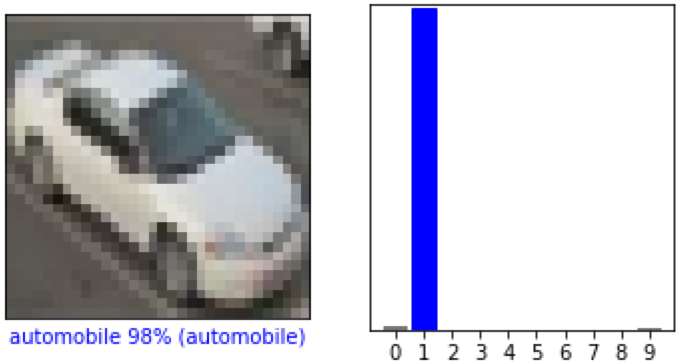


In [188…

```python
predictions_10 = model_10.predict(valid_images)
```

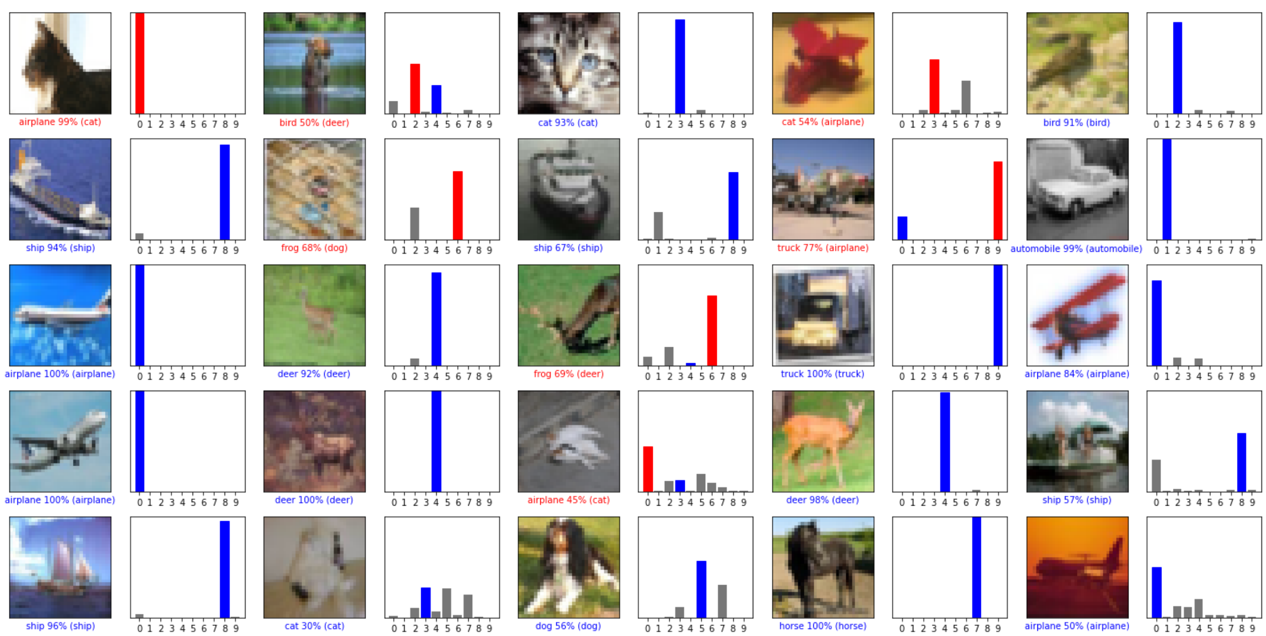
In [190…

```python
i = 35
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
#plot_image(i, predictions[i], test_labels, test_images)
plot_image(i, predictions_2[i], valid_labels, valid_images)
plt.subplot(1,2,2)
#plot_value_array(i, predictions[i],  test_labels)
plot_value_array(i, predictions_2[i],  valid_labels)
plt.show()
```



In [193…

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 5
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions_2[i], valid_labels, valid_images)
  # plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  # plot_value_array(i, predictions[i], test_labels)
  plot_value_array(i, predictions_2[i], valid_labels)
plt.tight_layout()
plt.show()
```



In [ ]:

Model 11 - Experiment with Kernel Regularization

Regularization was added as an attempt to improve on Model 10. Unfortunately, the addition of regularization on decreased the model's performance.

In [203…

```
model_11 = models.Sequential()
model_11.add(layers.Conv2D(32, (3, 3), padding='same', kernel_regularizer=tf.keras.regu
model_11.add(layers.Conv2D(32, (3, 3), padding ='same', kernel_regularizer=tf.keras.reg
model_11.add(BatchNormalization())
model_11.add(layers.Conv2D(32, (3, 3), padding ='same', kernel_regularizer=tf.keras.reg
model_11.add(BatchNormalization())
model_11.add(layers.Dropout(0.1))
model_11.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_11.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_regula
model_11.add(BatchNormalization())
model_11.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_regula
model_11.add(BatchNormalization())
model_11.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_regula
model_11.add(BatchNormalization())
model_11.add(layers.Dropout(0.1))
model_11.add(layers.MaxPooling2D((2, 2), strides=(2,2)))


model_11.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_regul
model_11.add(BatchNormalization())
model_11.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_regul
model_11.add(BatchNormalization())
model_11.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_regul
model_11.add(BatchNormalization())
model_11.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_regul
model_11.add(BatchNormalization())
model_11.add(layers.Dropout(0.1))
model_11.add(layers.MaxPooling2D((2, 2), strides=(2,2)))

model_11.add(layers.Flatten())
model_11.add(layers.Dense(128, activation='relu', kernel_regularizer=tf.keras.regulariz
model_11.add(BatchNormalization())
model_11.add(layers.Dropout(0.1))
model_11.add(layers.Dense(10, activation='softmax'))  # As noted above
```

In [204…

```
model_11.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), #
              #loss='sparse_categorical_crossentropy', # For TF1
              metrics=['accuracy'])
```

In [205…

```
history_11 = model_11.fit(train_images, train_labels, epochs=10,
                    validation_data=(valid_images, valid_labels))
```

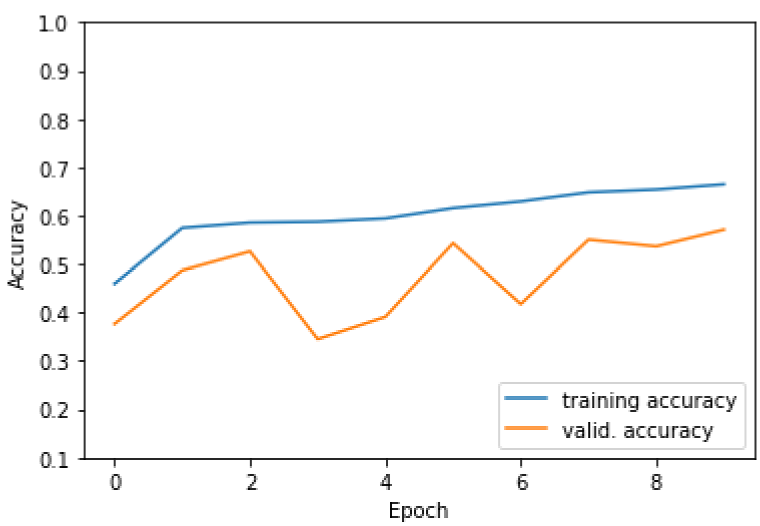
```
Epoch 1/10
1250/1250 [==============================] - 327s 260ms/step - loss: 4.7083 - accuracy:
0.4594 - val_loss: 2.8641 - val_accuracy: 0.3766
Epoch 2/10
1250/1250 [==============================] - 333s 266ms/step - loss: 1.8934 - accuracy:
0.5745 - val_loss: 2.1093 - val_accuracy: 0.4871
Epoch 3/10
1250/1250 [==============================] - 339s 271ms/step - loss: 1.7989 - accuracy:
0.5854 - val_loss: 1.8941 - val_accuracy: 0.5261
Epoch 4/10
1250/1250 [==============================] - 369s 295ms/step - loss: 1.7632 - accuracy:
0.5872 - val_loss: 2.7312 - val_accuracy: 0.3452
Epoch 5/10
```

```
1250/1250 [==============================] - 372s 297ms/step - loss: 1.7382 - accuracy:
0.5940 - val_loss: 2.1880 - val_accuracy: 0.3913
Epoch 6/10
1250/1250 [==============================] - 359s 287ms/step - loss: 1.6529 - accuracy:
0.6154 - val_loss: 1.9090 - val_accuracy: 0.5430
Epoch 7/10
1250/1250 [==============================] - 351s 281ms/step - loss: 1.5808 - accuracy:
0.6293 - val_loss: 2.0737 - val_accuracy: 0.4176
Epoch 8/10
1250/1250 [==============================] - 356s 284ms/step - loss: 1.4883 - accuracy:
0.6479 - val_loss: 1.7119 - val_accuracy: 0.5503
Epoch 9/10
1250/1250 [==============================] - 341s 273ms/step - loss: 1.4362 - accuracy:
0.6538 - val_loss: 1.8745 - val_accuracy: 0.5364
Epoch 10/10
1250/1250 [==============================] - 336s 269ms/step - loss: 1.3822 - accuracy:
0.6646 - val_loss: 1.6443 - val_accuracy: 0.5708
```

In [206…

```python
plt.plot(history_11.history['accuracy'], label='training accuracy') # For TF2
#plt.plot(history.history['acc'], label='training accuracy') # For TF1
plt.plot(history_11.history['val_accuracy'], label = 'valid. accuracy') # For TF2
#plt.plot(history.history['val_acc'], label = 'valid. accuracy') # For TF1
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.1, 1])
plt.legend(loc='lower right')

# Evaluate the learned model with validation set
valid_loss, valid_acc = model_11.evaluate(valid_images, valid_labels, verbose=2)
print ("valid_accuracy=%s, valid_loss=%s" % (valid_acc, valid_loss))
```

```
313/313 - 16s - loss: 1.6443 - accuracy: 0.5708 - 16s/epoch - 51ms/step
valid_accuracy=0.5708000063896179, valid_loss=1.644324779510498
```



In [ ]:

```python
test_loss, test_acc = model_11.evaluate(valid_images,  valid_labels)

print('\nTest accuracy:', test_acc)

print ('\n')
train_loss, train_acc = model_11.evaluate(train_images, train_labels, verbose = 2)
print('Training accuracy:', train_acc)
```

In [ ]:

In [200…

```
predictions = model_10.predict(test_images)

print(predictions)
```

```
[[9.8308057e-01 1.8212999e-05 2.5517668e-03 ... 8.5479184e-04
  7.4558267e-03 1.0308787e-03]
 [1.9771393e-02 4.8074129e-04 5.3020746e-01 ... 3.2225266e-02
  2.6721442e-02 1.1811379e-04]
 [1.1857492e-04 1.8295294e-03 1.2232403e-03 ... 6.4079337e-05
  4.3495267e-04 5.2719499e-04]
 ...
 [7.2197413e-06 1.5024254e-06 6.4147240e-04 ... 4.3253703e-03
  2.1635567e-06 7.2113835e-07]
 [9.4684534e-04 4.8566353e-06 5.7106060e-03 ... 4.6219058e-02
  8.2565202e-05 8.7258177e-06]
 [1.8457818e-03 1.5198733e-06 9.9797016e-01 ... 2.4859485e-06
  3.8984173e-05 7.5319730e-07]]
```

In [ ]:

```
import pandas as pd
```

In [209…

```
predict_DF = pd.DataFrame(predictions)
predict_DF = predict_DF.reset_index()
predict_DF.columns = ('id','cat0','cat1','cat2','cat3','cat4','cat5','cat6','cat7','cat
predict_DF['id'] = predict_DF['id'] + 1
predict_DF.to_csv('predictions.csv', index=False)
```

In [210…

```
test_images
```

Out[210…

```
array([[[[9.52876345e-06, 6.75456649e-06, 2.95512284e-06],
         [9.58907208e-06, 6.69425786e-06, 2.83450558e-06],
         [9.95092385e-06, 6.99580101e-06, 3.07574010e-06],
         ...,
         [8.26228223e-06, 5.72931979e-06, 2.17111066e-06],
         [7.59888731e-06, 5.48808528e-06, 2.17111066e-06],
         [6.99580101e-06, 5.12623350e-06, 1.99018477e-06]],

        [[9.16691167e-06, 6.75456649e-06, 3.07574010e-06],
         [9.10660304e-06, 6.63394924e-06, 2.41234518e-06],
         [9.58907208e-06, 6.87518375e-06, 2.71388832e-06],
         ...,
         [8.20197360e-06, 5.72931979e-06, 1.86956751e-06],
         [7.53857868e-06, 5.48808528e-06, 1.92987614e-06],
         [7.17672690e-06, 5.30715939e-06, 2.05049340e-06]],

        [[9.10660304e-06, 6.63394924e-06, 2.83450558e-06],
         [9.10660304e-06, 6.57364061e-06, 1.99018477e-06],
         [9.52876345e-06, 6.69425786e-06, 2.17111066e-06],
         ...,
         [8.38289949e-06, 5.91024568e-06, 2.05049340e-06],
         [7.84012182e-06, 5.72931979e-06, 2.05049340e-06],
         [7.23703553e-06, 5.36746802e-06, 1.99018477e-06]],

        ...,
```

```
 [[4.10098680e-06, 7.47827005e-06, 1.06746274e-05],
  [2.53296244e-06, 6.03086294e-06, 8.92567715e-06],
  [1.86956751e-06, 5.30715939e-06, 8.26228223e-06],
  ...,
  [2.29172792e-06, 5.84993705e-06, 8.80505989e-06],
  [7.84012182e-07, 3.85975228e-06, 6.51333198e-06],
  [2.41234518e-06, 5.12623350e-06, 7.65919594e-06]],

 [[3.67882639e-06, 6.99580101e-06, 1.01318497e-05],
  [2.95512284e-06, 6.15148020e-06, 8.92567715e-06],
  [2.11080203e-06, 5.12623350e-06, 7.96073908e-06],
  ...,
  [1.56802436e-06, 4.94530761e-06, 7.84012182e-06],
  [1.74895025e-06, 4.94530761e-06, 7.59888731e-06],
  [1.20617259e-06, 3.85975228e-06, 6.45302335e-06]],

 [[3.25666599e-06, 6.45302335e-06, 9.64938071e-06],
  [3.37728325e-06, 6.33240609e-06, 8.98598578e-06],
  [2.71388832e-06, 5.36746802e-06, 7.96073908e-06],
  ...,
  [1.44740711e-06, 4.64376446e-06, 7.47827005e-06],
  [2.05049340e-06, 5.06592487e-06, 7.77981319e-06],
  [1.26648122e-06, 4.04067817e-06, 6.63394924e-06]]],


[[[1.41725279e-05, 1.41725279e-05, 1.41725279e-05],
  [1.39312934e-05, 1.39312934e-05, 1.39312934e-05],
  [1.39916020e-05, 1.39916020e-05, 1.39916020e-05],
  ...,
  [1.40519107e-05, 1.40519107e-05, 1.40519107e-05],
  [1.40519107e-05, 1.40519107e-05, 1.40519107e-05],
  [1.39916020e-05, 1.39916020e-05, 1.39916020e-05]],

 [[1.43534538e-05, 1.43534538e-05, 1.43534538e-05],
  [1.41725279e-05, 1.41725279e-05, 1.41725279e-05],
  [1.41725279e-05, 1.41725279e-05, 1.41725279e-05],
  ...,
  [1.42328365e-05, 1.42328365e-05, 1.42328365e-05],
  [1.42328365e-05, 1.42328365e-05, 1.42328365e-05],
  [1.41725279e-05, 1.41725279e-05, 1.41725279e-05]],

 [[1.42931452e-05, 1.42931452e-05, 1.42931452e-05],
  [1.41122193e-05, 1.41122193e-05, 1.41122193e-05],
  [1.41122193e-05, 1.41122193e-05, 1.41122193e-05],
  ...,
  [1.41725279e-05, 1.41725279e-05, 1.41725279e-05],
  [1.41725279e-05, 1.41725279e-05, 1.41725279e-05],
  [1.41122193e-05, 1.41122193e-05, 1.41122193e-05]],

 ...,

 [[5.24685076e-06, 5.97055431e-06, 5.36746802e-06],
  [2.59327106e-06, 3.07574010e-06, 2.23141929e-06],
  [1.14586396e-06, 1.38709848e-06, 6.63394924e-07],
  ...,
  [1.01921584e-05, 1.10967878e-05, 1.07952447e-05],
  [1.09761706e-05, 1.18808000e-05, 1.16395655e-05],
  [1.13380223e-05, 1.21823431e-05, 1.21220345e-05]],
```

```
 [[4.94530761e-06, 5.78962842e-06, 4.94530761e-06],
  [2.77419695e-06, 3.43759188e-06, 2.17111066e-06],
  [2.17111066e-06, 2.65357969e-06, 1.32678985e-06],
  ...,
  [1.04937015e-05, 1.13983310e-05, 1.10364792e-05],
  [1.11570964e-05, 1.20617259e-05, 1.18204914e-05],
  [1.12777137e-05, 1.21823431e-05, 1.20617259e-05]],

 [[5.12623350e-06, 6.09117157e-06, 5.00561624e-06],
  [3.73913502e-06, 4.52314721e-06, 2.89481421e-06],
  [3.49790051e-06, 4.04067817e-06, 2.29172792e-06],
  ...,
  [1.01318497e-05, 1.10364792e-05, 1.07349360e-05],
  [1.08555533e-05, 1.17601827e-05, 1.15189482e-05],
  [1.12174051e-05, 1.20617259e-05, 1.20014173e-05]]],


[[[9.52876345e-06, 1.14586396e-05, 1.33885157e-05],
  [9.52876345e-06, 1.12777137e-05, 1.31472812e-05],
  [8.38289949e-06, 1.00112325e-05, 1.16998741e-05],
  ...,
  [1.37503675e-05, 1.39312934e-05, 1.41122193e-05],
  [1.42931452e-05, 1.44137624e-05, 1.46549969e-05],
  [1.43534538e-05, 1.45343797e-05, 1.48359228e-05]],

 [[1.02524670e-05, 1.20617259e-05, 1.38106761e-05],
  [1.03730843e-05, 1.20014173e-05, 1.36297502e-05],
  [9.10660304e-06, 1.06143188e-05, 1.21220345e-05],
  ...,
  [1.39916020e-05, 1.39916020e-05, 1.42328365e-05],
  [1.48359228e-05, 1.48359228e-05, 1.50771574e-05],
  [1.48359228e-05, 1.48962315e-05, 1.51374660e-05]],

 [[1.04937015e-05, 1.21220345e-05, 1.35694416e-05],
  [1.06143188e-05, 1.20617259e-05, 1.33885157e-05],
  [9.46845482e-06, 1.07952447e-05, 1.20014173e-05],
  ...,
  [1.38709848e-05, 1.38106761e-05, 1.39916020e-05],
  [1.50771574e-05, 1.50168487e-05, 1.51374660e-05],
  [1.47756142e-05, 1.47153056e-05, 1.48962315e-05]],

 ...,

 [[1.86956751e-06, 2.41234518e-06, 2.71388832e-06],
  [1.80925888e-06, 2.35203655e-06, 2.65357969e-06],
  [1.56802436e-06, 2.11080203e-06, 2.41234518e-06],
  ...,
  [2.23141929e-06, 2.41234518e-06, 2.77419695e-06],
  [5.42777665e-07, 7.84012182e-07, 8.44320812e-07],
  [2.41234518e-07, 4.22160406e-07, 3.01543147e-07]],

 [[1.38709848e-06, 2.05049340e-06, 2.35203655e-06],
  [1.62833299e-06, 2.29172792e-06, 2.59327106e-06],
  [1.50771574e-06, 2.17111066e-06, 2.47265381e-06],
  ...,
  [1.14586396e-06, 1.20617259e-06, 1.44740711e-06],
  [2.41234518e-07, 3.61851776e-07, 1.80925888e-07],
  [3.01543147e-07, 4.22160406e-07, 1.80925888e-07]],

 [[1.68864162e-06, 2.47265381e-06, 2.83450558e-06],
```

```
  [1.80925888e-06, 2.59327106e-06, 3.01543147e-06],
  [1.92987614e-06, 2.71388832e-06, 3.13604873e-06],
  ...,
  [3.01543147e-07, 3.61851776e-07, 4.82469035e-07],
  [2.41234518e-07, 3.01543147e-07, 1.80925888e-07],
  [4.22160406e-07, 4.82469035e-07, 4.22160406e-07]]],


...,


[[[1.20617259e-06, 9.04629441e-07, 7.23703553e-07],
  [1.14586396e-06, 8.44320812e-07, 6.63394924e-07],
  [9.04629441e-07, 8.44320812e-07, 6.63394924e-07],
  ...,
  [6.03086294e-07, 5.42777665e-07, 4.22160406e-07],
  [7.23703553e-07, 6.63394924e-07, 5.42777665e-07],
  [7.84012182e-07, 7.23703553e-07, 6.03086294e-07]],

 [[1.26648122e-06, 9.64938071e-07, 7.84012182e-07],
  [1.20617259e-06, 9.64938071e-07, 7.84012182e-07],
  [1.08555533e-06, 1.02524670e-06, 7.23703553e-07],
  ...,
  [6.03086294e-07, 5.42777665e-07, 4.22160406e-07],
  [6.03086294e-07, 5.42777665e-07, 4.22160406e-07],
  [7.23703553e-07, 6.63394924e-07, 5.42777665e-07]],

 [[1.26648122e-06, 9.64938071e-07, 7.84012182e-07],
  [1.26648122e-06, 1.02524670e-06, 7.23703553e-07],
  [1.20617259e-06, 1.08555533e-06, 6.63394924e-07],
  ...,
  [7.23703553e-07, 6.63394924e-07, 5.42777665e-07],
  [7.23703553e-07, 6.63394924e-07, 5.42777665e-07],
  [7.84012182e-07, 7.23703553e-07, 6.03086294e-07]],

 ...,

 [[1.99018477e-06, 1.50771574e-06, 7.84012182e-07],
  [2.05049340e-06, 1.56802436e-06, 9.04629441e-07],
  [2.05049340e-06, 1.56802436e-06, 9.04629441e-07],
  ...,
  [1.68864162e-06, 1.50771574e-06, 3.13604873e-06],
  [1.74895025e-06, 1.50771574e-06, 3.49790051e-06],
  [1.38709848e-06, 1.20617259e-06, 2.53296244e-06]],

 [[1.99018477e-06, 1.50771574e-06, 8.44320812e-07],
  [2.05049340e-06, 1.56802436e-06, 9.04629441e-07],
  [2.05049340e-06, 1.56802436e-06, 9.04629441e-07],
  ...,
  [1.62833299e-06, 1.44740711e-06, 3.13604873e-06],
  [1.62833299e-06, 1.44740711e-06, 3.37728325e-06],
  [1.50771574e-06, 1.32678985e-06, 2.83450558e-06]],

 [[1.86956751e-06, 1.38709848e-06, 7.23703553e-07],
  [1.92987614e-06, 1.44740711e-06, 7.84012182e-07],
  [1.99018477e-06, 1.50771574e-06, 8.44320812e-07],
  ...,
  [1.44740711e-06, 1.38709848e-06, 3.01543147e-06],
  [1.56802436e-06, 1.38709848e-06, 3.19635736e-06],
  [1.50771574e-06, 1.20617259e-06, 2.83450558e-06]]],
```

```
[[[1.50771574e-06, 2.41234518e-06, 7.23703553e-07],
  [9.04629441e-07, 2.17111066e-06, 1.80925888e-07],
  [1.38709848e-06, 2.47265381e-06, 1.08555533e-06],
  ...,
  [3.67882639e-06, 4.94530761e-06, 4.70407309e-06],
  [5.54839391e-06, 6.81487512e-06, 6.75456649e-06],
  [4.52314721e-06, 5.36746802e-06, 5.54839391e-06]],

 [[7.23703553e-07, 1.50771574e-06, 3.61851776e-07],
  [1.20617259e-06, 2.23141929e-06, 4.22160406e-07],
  [1.44740711e-06, 2.17111066e-06, 9.04629441e-07],
  ...,
  [6.93549238e-06, 8.08135634e-06, 8.32259086e-06],
  [8.98598578e-06, 1.01318497e-05, 1.06746274e-05],
  [6.27209746e-06, 7.05610964e-06, 7.90043045e-06]],

 [[7.23703553e-07, 1.50771574e-06, 6.63394924e-07],
  [9.04629441e-07, 1.74895025e-06, 3.61851776e-07],
  [2.05049340e-06, 2.41234518e-06, 1.44740711e-06],
  ...,
  [9.28752893e-06, 1.03730843e-05, 1.09761706e-05],
  [9.46845482e-06, 1.05540101e-05, 1.15792568e-05],
  [6.99580101e-06, 7.77981319e-06, 9.10660304e-06]],

 ...,

 [[6.03086294e-06, 7.77981319e-06, 4.88499898e-06],
  [6.21178883e-06, 7.96073908e-06, 5.06592487e-06],
  [6.27209746e-06, 8.08135634e-06, 5.18654213e-06],
  ...,
  [5.84993705e-06, 7.71950456e-06, 5.06592487e-06],
  [5.91024568e-06, 7.59888731e-06, 5.06592487e-06],
  [5.48808528e-06, 7.29734416e-06, 4.76438172e-06]],

 [[6.21178883e-06, 7.96073908e-06, 5.00561624e-06],
  [6.27209746e-06, 7.90043045e-06, 5.00561624e-06],
  [6.45302335e-06, 8.14166497e-06, 5.24685076e-06],
  ...,
  [6.09117157e-06, 7.96073908e-06, 5.24685076e-06],
  [5.97055431e-06, 7.65919594e-06, 5.06592487e-06],
  [5.54839391e-06, 7.29734416e-06, 4.76438172e-06]],

 [[5.72931979e-06, 7.59888731e-06, 4.70407309e-06],
  [5.72931979e-06, 7.41796142e-06, 4.58345584e-06],
  [6.09117157e-06, 7.71950456e-06, 4.88499898e-06],
  ...,
  [5.60870254e-06, 7.47827005e-06, 4.82469035e-06],
  [5.72931979e-06, 7.41796142e-06, 4.88499898e-06],
  [5.54839391e-06, 7.23703553e-06, 4.82469035e-06]]],


[[[4.40252995e-06, 4.70407309e-06, 4.52314721e-06],
  [5.91024568e-06, 6.21178883e-06, 6.81487512e-06],
  [5.97055431e-06, 6.39271472e-06, 6.87518375e-06],
  ...,
  [8.14166497e-06, 9.04629441e-06, 9.16691167e-06],
  [8.14166497e-06, 8.98598578e-06, 9.28752893e-06],
  [1.22426518e-05, 1.29663553e-05, 1.34488244e-05]],
```

```
[[4.16129543e-06, 4.40252995e-06, 4.22160406e-06],
 [5.06592487e-06, 5.36746802e-06, 5.84993705e-06],
 [4.10098680e-06, 4.52314721e-06, 4.88499898e-06],
 ...,
 [5.12623350e-06, 5.72931979e-06, 5.36746802e-06],
 [4.28191269e-06, 4.94530761e-06, 4.82469035e-06],
 [7.23703553e-06, 8.02104771e-06, 8.14166497e-06]],

[[4.16129543e-06, 4.40252995e-06, 4.22160406e-06],
 [5.42777665e-06, 5.72931979e-06, 6.03086294e-06],
 [3.73913502e-06, 4.28191269e-06, 4.46283858e-06],
 ...,
 [4.46283858e-06, 4.88499898e-06, 4.22160406e-06],
 [3.19635736e-06, 3.73913502e-06, 3.25666599e-06],
 [3.73913502e-06, 4.46283858e-06, 4.16129543e-06]],

...,

[[7.41796142e-06, 7.71950456e-06, 5.78962842e-06],
 [7.96073908e-06, 7.96073908e-06, 6.15148020e-06],
 [7.77981319e-06, 7.71950456e-06, 6.03086294e-06],
 ...,
 [6.51333198e-06, 6.45302335e-06, 5.30715939e-06],
 [3.73913502e-06, 3.61851776e-06, 3.31697462e-06],
 [1.62833299e-06, 1.62833299e-06, 1.68864162e-06]],

[[6.93549238e-06, 7.29734416e-06, 5.48808528e-06],
 [7.41796142e-06, 7.47827005e-06, 5.72931979e-06],
 [7.77981319e-06, 7.59888731e-06, 5.97055431e-06],
 ...,
 [6.93549238e-06, 6.99580101e-06, 5.66901116e-06],
 [3.98036954e-06, 3.92006091e-06, 3.55820914e-06],
 [1.62833299e-06, 1.62833299e-06, 1.62833299e-06]],

[[6.99580101e-06, 7.23703553e-06, 5.42777665e-06],
 [7.29734416e-06, 7.35765279e-06, 5.66901116e-06],
 [7.77981319e-06, 7.71950456e-06, 6.09117157e-06],
 ...,
 [6.99580101e-06, 6.93549238e-06, 5.66901116e-06],
 [4.10098680e-06, 3.92006091e-06, 3.49790051e-06],
 [1.62833299e-06, 1.56802436e-06, 1.56802436e-06]]]])
```

In [ ]: