ANN vs CNN for Regression and Image Categorization

by Eric Tran

ANNs are a very useful tool for AI. In this project I want to show how to use a regular ANN to predict the price of a diamond based on its dimensions, carat, color, cut, and clarity. I will also show how to use a CNN to identify emotions based on a picture of a face.

First, let's turn off warnings, and import our necessary libraries.

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
# import system libraries
import os
import shutil
import datetime

# import pandas, numpy, and visualization libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# import keras and tensorflow
import keras
import tensorflow as tf
tf.config.run_functions_eagerly(True) # to avoid tf.function-decorated functions creating 

# import preprocessing and postprocessing libraries
from tensorboard.plugins.hparams import api as hp
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from tensorboard.plugins.hparams import api as hp

# import model types
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
import xgboost as xgb

# import model utlities
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten

# import libraries for confusion matrix
from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools
```

ANN

Next we will load the dataset, which can be found at [https://www.kaggle.com/datasets/joebeachcapital/diamonds (https://www.kaggle.com/datasets/joebeachcapital/diamonds)](https://www.kaggle.com/datasets/joebeachcapital/diamonds)

In [36]:
```
diamonds = pd.read_csv('diamonds_dataset/diamonds.csv')
```

Data Exploration

Now we will start exploring the data.

In [37]:
```
diamonds.head()
```

Out[37]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

We can see that cut, color, and clarity are not numbers but categories, so we will one hot encode it later.

In [ ]:
```
diamonds.shape
```

In [39]:
```
diamonds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

In [40]:
```
diamonds.describe()
```

Out[40]:

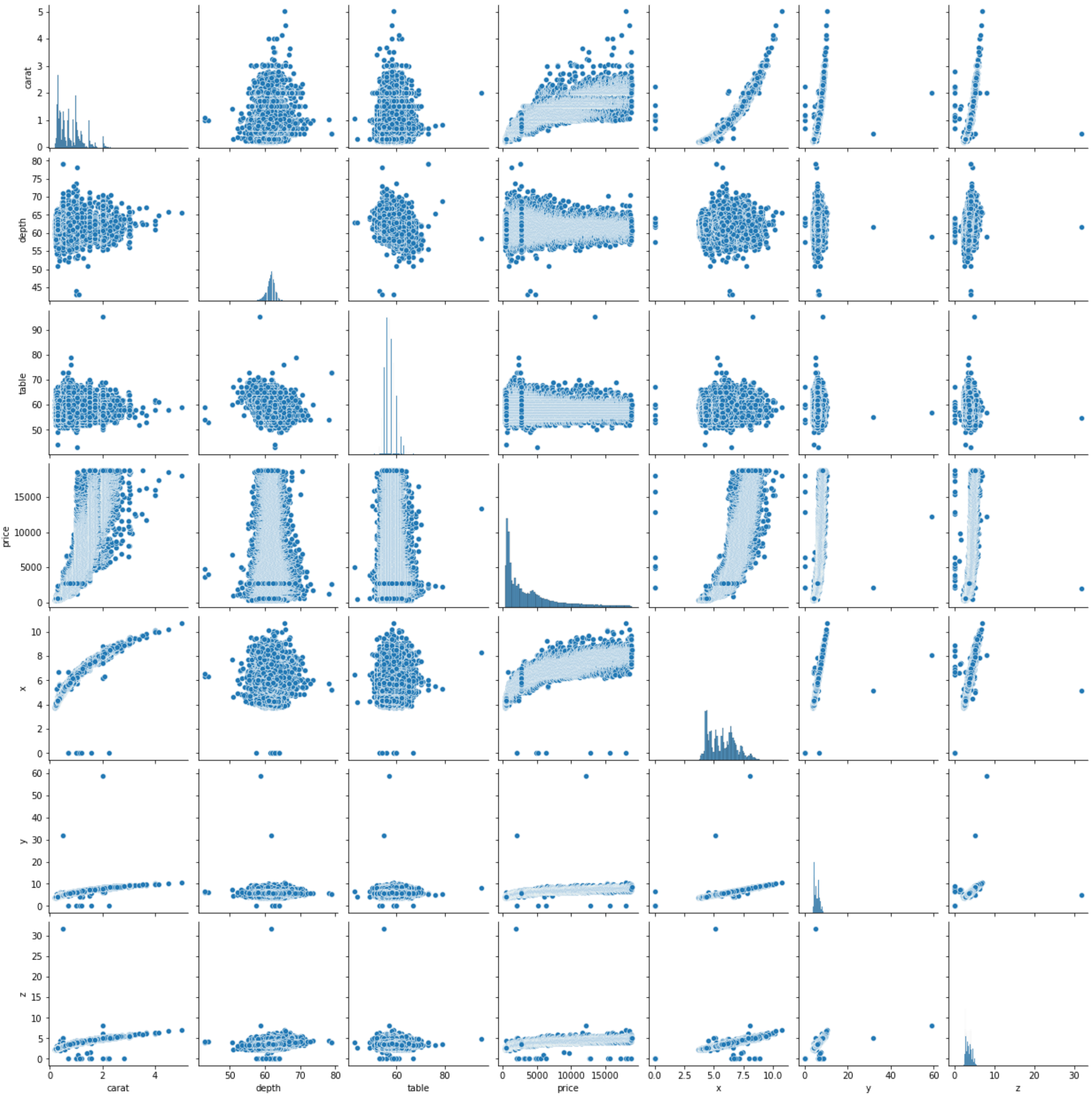
	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

In [41]:
```
diamonds.columns
```

Out[41]:
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z'],
      dtype='object')
```

In [42]:
```
sns.pairplot(diamonds)
```

Out[42]: <seaborn.axisgrid.PairGrid at 0x24b21148040>

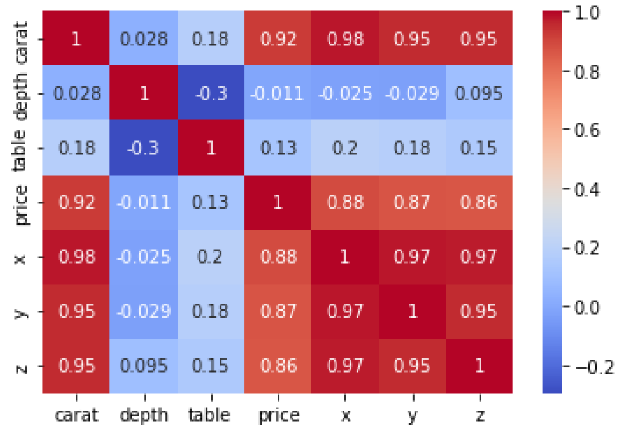


From the pairplots, we can see that there is strong correlations between carat, x, and price, which makes sense since a higher x means a bigger carat, and we all know the more carats the higher the more expensive a diamond is.

We can also see that in the heatmap below.

In [43]:
```
corr_matrix = diamonds.select_dtypes(exclude=['object']).corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

Out[43]: <AxesSubplot:>



Data Cleaning

The first thing we will do is to one hot encode the columns which were objects and turn them into separate columns for each category in the original columns.

In [44]:
```
# Save Object columns to variable
object_columns = diamonds.select_dtypes(include=['object'])

# Create a OneHotEncoder object
encoder = OneHotEncoder(sparse=False, drop='first')

# Fit the encoder to the 'object' type columns data
encoder.fit(object_columns)

# Perform one-hot encoding to obtain the new columns
encoded_data = encoder.transform(object_columns)

# Create a new DataFrame with the encoded columns
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(object_colum

# Add the encoded DataFrame to your original DataFrame
diamonds = pd.concat([diamonds, encoded_df], axis=1)

# Drop object columns since they have been encoded
diamonds = diamonds.drop(object_columns.columns, axis=1)
```

In [45]:
```
diamonds.head()
```

Out[45]:

	carat	depth	table	price	x	y	z	cut_Good	cut_Ideal	cut_Premium	...	color_H	color_I	color_J
0	0.23	61.5	55.0	326	3.95	3.98	2.43	0.0	1.0	0.0	...	0.0	0.0	0.0
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0.0	0.0	1.0	...	0.0	0.0	0.0
2	0.23	56.9	65.0	327	4.05	4.07	2.31	1.0	0.0	0.0	...	0.0	0.0	0.0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0.0	0.0	1.0	...	0.0	1.0	0.0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	1.0	0.0	0.0	...	0.0	0.0	1.0

5 rows × 24 columns

In [46]:
```
diamonds.columns
```

Out[46]:
```
Index(['carat', 'depth', 'table', 'price', 'x', 'y', 'z', 'cut_Good',
       'cut_Ideal', 'cut_Premium', 'cut_Very Good', 'color_E', 'color_F',
       'color_G', 'color_H', 'color_I', 'color_J', 'clarity_IF', 'clarity_SI1',
       'clarity_SI2', 'clarity_VS1', 'clarity_VS2', 'clarity_VVS1',
       'clarity_VVS2'],
      dtype='object')
```

Next, we will split the data into train and test with train_test_split so we can fit it to our models later.

In [47]:
```

X = diamonds.loc[:, diamonds.columns != 'price']
y = diamonds['price']
```

In [48]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=62)
```

We will also scale the X values to avoid issues.

In [49]:
```
sc = StandardScaler()
```

In [50]:
```
X_train_scaled = sc.fit(X_train)
X_test_scaled = sc.fit(X_test)
```

Finding the Best Model

We will now make a list of models, and we will fit and test each model in a for loop to determine which model performed best.

In [51]:

```
models = [LinearRegression(),
          DecisionTreeRegressor(max_depth=10),
          RandomForestRegressor(n_estimators=100, random_state=42),
          xgb.XGBRegressor(n_estimators=100, random_state=42),
          AdaBoostRegressor(n_estimators=100, random_state=42)]

# For loop iterates through the models and fits the data before testing and displaying metr
for model in models:
    # Fit the model to the training data
    model.fit(X_train, y_train)

    # Make predictions on training and test data
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Calculate Mean Squared Error (MSE)
    mse_test = mean_squared_error(y_test, y_test_pred)

    # Calculate R-squared on training and test data
    r2_train = model.score(X_train, y_train)
    r2_test = model.score(X_test, y_test)

    # Print the metrics
    print(f'Model: {type(model).__name__}')
    print(f'Mean Squared Error (MSE): {mse_test:.2f}')
    print(f'R-squared (R²) on training data: {r2_train:.2f}%')
    print(f'R-squared (R²) on test data: {r2_test:.2f}%')
    print('-------------------------------------')
```

```
Model: LinearRegression
Mean Squared Error (MSE): 1305785.21
R-squared (R²) on training data: 0.92%
R-squared (R²) on test data: 0.92%
-------------------------------------
Model: DecisionTreeRegressor
Mean Squared Error (MSE): 772024.58
R-squared (R²) on training data: 0.96%
R-squared (R²) on test data: 0.95%
-------------------------------------
Model: RandomForestRegressor
Mean Squared Error (MSE): 405208.53
R-squared (R²) on training data: 1.00%
R-squared (R²) on test data: 0.97%
-------------------------------------
Model: XGBRegressor
Mean Squared Error (MSE): 343196.38
R-squared (R²) on training data: 0.99%
R-squared (R²) on test data: 0.98%
-------------------------------------
Model: AdaBoostRegressor
Mean Squared Error (MSE): 2128357.13
R-squared (R²) on training data: 0.87%
R-squared (R²) on test data: 0.87%
-------------------------------------
```

In [ ]:
```
# Look at confusion matrix
#Note, this code is taken straight from the SKLEARN website, an nice way of viewing confus
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Observation')
    plt.xlabel('Prediction')
```

In [ ]:
```
# Predict the values from the validation dataset
Y_pred = model.predict(X_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred, axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

It looks like XGBRegressor did the best, followed by RandomForestRegressor, then DecisionTreeRegressor, then LinearRegression. AdaBoostRegressor performed the worst in this case.

CNN

Now we will use CNN to determine emotion from a picture. Although I would normally combine the train and test datasets and resplit myself with train_test_split, I decided to try it as is to practice other ways of doing it.

These images are 48x48.

The dataset can be found at https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer

In [52]:
```
train_dir = "./emotions_dataset/train" # save the path with training images
test_dir = "./emotions_dataset/test"   # save the path with testing images
```

In [53]:
```
classes=os.listdir(train_dir+'/')
classes
```

Out[53]: ['angry', 'disgusted', 'fearful', 'happy', 'neutral', 'sad', 'surprised']

In order to load the pictures, I will use ImageDataGenerator to load and split the data into a train variable and a validation variable. I am using this instead of test_train_split because all the images are pre-categorized by their folder locations.

In [54]:
```
"""
Data Augmentation
--------------------------
rotation_range = rotates the image with the amount of degrees we provide
width_shift_range = shifts the image randomly to the right or left along the width of the i
height_shift range = shifts image randomly to up or below along the height of the image
horizontal_flip = flips the image horizontally
rescale = to scale down the pixel values in our image between 0 and 1
zoom_range = applies random zoom to our object
validation_split = reserves some images to be used for validation purpose
"""

train_datagen = ImageDataGenerator(#rotation_range = 180,
                                         width_shift_range = 0.1,
                                         height_shift_range = 0.1,
                                         horizontal_flip = True,
                                         rescale = 1./255,
                                         #zoom_range = 0.2,
                                         validation_split = 0.3
                                        )
validation_datagen = ImageDataGenerator(rescale = 1./255,
                                         validation_split = 0.3)
```

I will now use the instances of ImageDataGenerator to make the train and test datasets, although they will not be in the X_train, X_test, y_train, y_test like we normally use, but rather both X_train and y_train will be in one, and X_test and y_test in the other.

In [55]:
```
train_generator = train_datagen.flow_from_directory(directory = train_dir,
                                                    target_size = (48,48),
                                                    batch_size = 64,
                                                    color_mode = "grayscale",
                                                    class_mode = "categorical",
                                                    subset = "training"
                                                   )
validation_generator = validation_datagen.flow_from_directory( directory = test_dir,
                                                              target_size = (48,48),
                                                              batch_size = 64,
                                                              color_mode = "grayscale",
                                                              class_mode = "categorical",
                                                              subset = "validation"
                                                             )
```

```
Found 20099 images belonging to 7 classes.
Found 2151 images belonging to 7 classes.
```

I couldn't figure out how to print the first 50 images from the generator, so I will split it into x train and y train.

In [ ]:
```
# DO NOT RUN THIS CELL! IT FROZE MY PUNY LAPTOP! TRY ON VM OR LAPTOP WITH GPU!
x_train_split = []
y_train_split = []
for x, y in train_generator:
    x_train_split.append(x)
    y_train_split.append(y)
```

In [ ]:
```
plt.figure(figsize=(15,15)) # size of each image
for i in range(50): # iterate for first 50 images
    plt.subplot(10,5,i+1) #  10 rows, 5 columns, index
    plt.xticks([]) # get rid of ticks on axis
    plt.yticks([]) # ditto
    plt.grid(False) # not sure necessary
    #plt.imshow(X_train[i],cmap=plt.cm.binary) # read image as binary
    plt.imshow(x_train_split[i],cmap='Greys') # different version
    plt.xlabel(classes[y_train_split[i]]) # label each item on x label
plt.tight_layout()
plt.show()
```

Next is to make our CNN model. I made it have 3 convolution layers, one being the input, and 3 dense layers, one being the output. I also applied MaxPooling2D and Dropout between several of the layers.

In [57]:
```
cnn_model = Sequential()
cnn_model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu', input_shape = (48,48,1)
cnn_model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2,2)))
cnn_model.add(Dropout(0.3))
cnn_model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2,2)))
cnn_model.add(Dropout(0.3))
cnn_model.add(Flatten())
cnn_model.add(Dense(128, activation = 'relu'))
cnn_model.add(Dropout(0.3))
cnn_model.add(Dense(128, activation = 'relu'))
cnn_model.add(Dropout(0.4))
cnn_model.add(Dense(7, activation='softmax'))
cnn_model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=['accuracy'
```

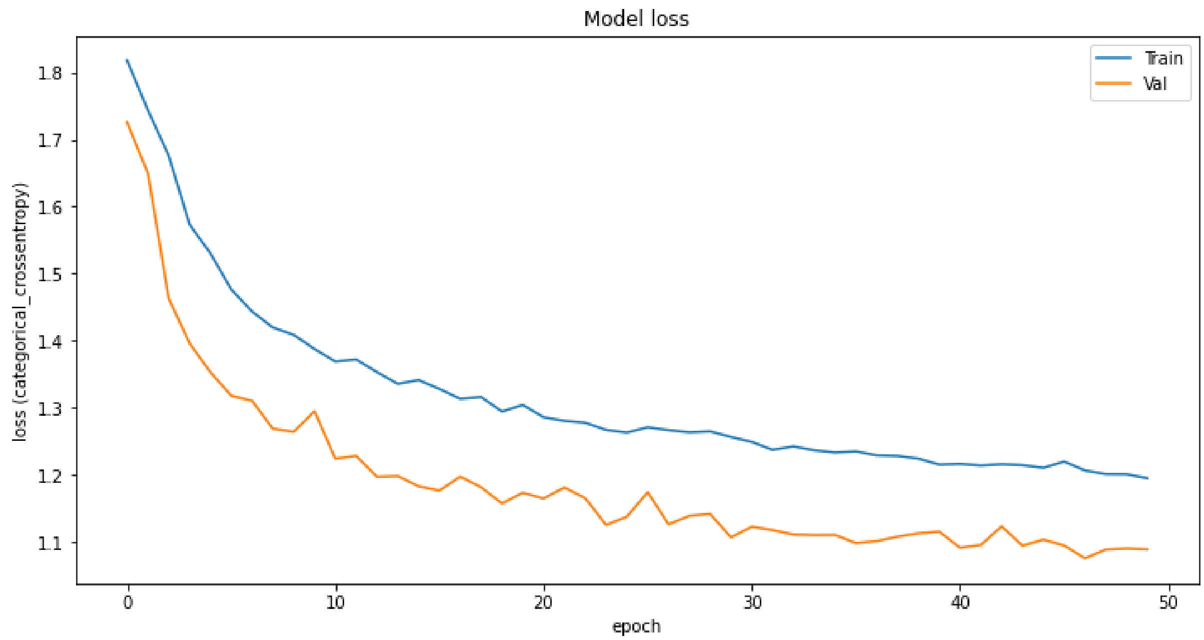
In [58]:
```
cnn_model.fit(train_generator, batch_size = 32, epochs = 50, verbose =1, validation_data=(
```

```
Epoch 1/50
315/315 [==============================] - 141s 449ms/step - loss: 1.8178 - accuracy:
0.2468 - val_loss: 1.7256 - val_accuracy: 0.2920
Epoch 2/50
315/315 [==============================] - 129s 409ms/step - loss: 1.7435 - accuracy:
0.2858 - val_loss: 1.6496 - val_accuracy: 0.3491
Epoch 3/50
315/315 [==============================] - 137s 436ms/step - loss: 1.6760 - accuracy:
0.3319 - val_loss: 1.4632 - val_accuracy: 0.4449
Epoch 4/50
315/315 [==============================] - 132s 419ms/step - loss: 1.5733 - accuracy:
0.3818 - val_loss: 1.3963 - val_accuracy: 0.4593
Epoch 5/50
315/315 [==============================] - 332s 1s/step - loss: 1.5303 - accuracy: 0.4
040 - val_loss: 1.3532 - val_accuracy: 0.4840
Epoch 6/50
315/315 [==============================] - 150s 477ms/step - loss: 1.4767 - accuracy:
0.4297 - val_loss: 1.3184 - val_accuracy: 0.5058
Epoch 7/50
```

Now we will plot the loss and show the model's best loss and accuracy values.

In [60]:
```
plt.figure(figsize=(12,6))
plt.plot(cnn_model.history.history['loss'][:])
plt.plot(cnn_model.history.history['val_loss'][:])
plt.title('Model loss')
plt.xlabel('epoch')
plt.ylabel('loss (categorical_crossentropy)')
plt.legend(['Train', 'Val'], loc='upper right')
```

Out[60]: <matplotlib.legend.Legend at 0x24b76a66f40>



In [61]:
```
loss, accuracy = cnn_model.evaluate(validation_generator)
print(f'loss: {loss}')
print(f'accuracy: {accuracy}')
```

```
34/34 [==============================] - 3s 76ms/step - loss: 1.0894 - accuracy: 0.6034
loss: 1.0893890857696533
accuracy: 0.6034402847290039
```

Accuracy is only 60%, which is worse than I expected, but to be honest I didn't think some of the images were properly categorized.

In [ ]:

Hyperparameters and Tensorboard

Now we will use Tensorboard to visualize and see which hyperparameter values work best. We will start by loading the tensorboard extension and clearing any previous logs as to not run into issues.

In [1]:
```
%load_ext tensorboard
```

In [5]:
```
folder_path = "logs/"

# Check if the folder exists before attempting to delete it
if os.path.exists(folder_path):
    # Remove the folder and its contents recursively
    shutil.rmtree(folder_path)
    # rm -rf ./logs/  # MAC users
    print(f"The folder '{folder_path}' has been deleted.")
else:
    print(f"The folder '{folder_path}' does not exist.")
```

```
The folder 'logs/' does not exist.
```

In [ ]:
```
# Delete the ".tensorboard-info" directory

folder_path = "C:/Users/angel/AppData/Local/Temp/.tensorboard-info/"

# Check if the folder exists before attempting to delete it
if os.path.exists(folder_path):
    # Remove the folder and its contents recursively
    shutil.rmtree(folder_path)
    print(f"The folder '{folder_path}' has been deleted.")
else:
    print(f"The folder '{folder_path}' does not exist.")
```

Now we will make logs for hyperparameter tuning, and define a function to test all the values to see which works best.

In [ ]:
```
HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([8, 16, 32, 64])) # values for number of
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2, 0.3, 0.4)) # values for dropou
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd'])) # values for optimizer

METRIC_ACCURACY = 'accuracy'

with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
  hp.hparams_config(
    hparams=[HP_NUM_UNITS, HP_DROPOUT, HP_OPTIMIZER],
    metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
  )
```

In [ ]:
```
def train_test_model(hparams):
  model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(hparams[HP_NUM_UNITS], activation='relu'),
    tf.keras.layers.Dropout(hparams[HP_DROPOUT]),
    tf.keras.layers.Dense(hparams[HP_NUM_UNITS], activation='relu'),
    tf.keras.layers.Dropout(hparams[HP_DROPOUT]),
    tf.keras.layers.Dense(10, activation='softmax'),
  ])
  model.compile(
      optimizer=hparams[HP_OPTIMIZER],
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'],
  )

  model.fit(x_train_scaled, y_train, epochs=10)
  _, accuracy = model.evaluate(x_test_scaled, y_test)
  return accuracy
```

In [ ]:
```
def run(run_dir, hparams):
  with tf.summary.create_file_writer(run_dir).as_default():
    hp.hparams(hparams)  # record the values used in this trial
    accuracy = train_test_model(hparams)
    tf.summary.scalar(METRIC_ACCURACY, accuracy, step=1)
```

In [ ]:
```
session_num = 0

for num_units in HP_NUM_UNITS.domain.values:
  for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_value):
    for optimizer in HP_OPTIMIZER.domain.values:
      hparams = {
          HP_NUM_UNITS: num_units,
          HP_DROPOUT: dropout_rate,
          HP_OPTIMIZER: optimizer,
      }
      run_name = "run-%d" % session_num
      print('--- Starting trial: %s' % run_name)
      print({h.name: hparams[h] for h in hparams})
      run('logs/hparam_tuning/' + run_name, hparams)
      session_num += 1
```

In [ ]:
```
%tensorboard --logdir logs/hparam_tuning
```

Conclusions

It looks like I have to tune my hyperparameters more. It also looks like I need to do this one a VM or my other laptop with a GPU as my computer crashed multiple times. Given more time, I would switch computers and test more.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: